

Notation

Un **diagramme d'activité** peut être divisé visuellement en couloirs, chacun séparé de ses couloirs de nage voisins par des lignes verticales continues (**Figure 13-176**). Chaque couloir de nage représente une responsabilité de haut niveau pour une partie de l'activité globale, qui peut sera finalement implémenté par un ou plusieurs objets. L'ordre relatif des

Le terme « couloirs » n'a pas de signification sémantique, mais peut indiquer une certaine affinité avec le monde réel. Chaque état d'activité est associé à un couloir et y est représenté visuellement. Les transitions peuvent traverser des voies ; le tracé d'une transition n'a aucune importance. chemin.

Les couloirs de nage n'étant que des partitions en catégories arbitraires, ils peuvent être indiqués par d'autres moyens si une disposition géométrique en régions est impraticable. Les possibilités incluent l'utilisation de la couleur ou simplement l'utilisation de valeurs étiquetées pour afficher partition.

état de synchronisation

Un **état** spécial qui permet la synchronisation du contrôle entre deux régions concurrentes dans une machine à états.

Voir aussi **transition complexe, état composite, bifurcation, jonction, machine à états, transition**.

Sémantique

Un **état composite** peut être constitué de plusieurs régions concurrentes, chacune possédant son propre ensemble séquentiel d'états. Lorsqu'un état composite concurrent est atteint, chaque région concurrente devient active. Il existe un seul **flux** de contrôle dans chaque région concurrente. Chaque région s'exécute indépendamment des autres (c'est le principe de la concurrence). Cependant, il arrive que le contrôle doive être synchronisé entre plusieurs régions concurrentes. régions. Une approche consiste à imposer une **condition de garde** lors de la transition dans une région. Cela dépend de l'état d'une autre région. Cela peut être utile pour l'exclusion mutuelle, notamment en cas de partage de ressources, mais cela ne prend pas en compte les situations où une activité dans une région a des conséquences ultérieures dans une autre région (même si (la première région peut alors passer à la suivante). Pour saisir cette dernière situation, les états de synchronisation peuvent être utilisés.

Un état de synchronisation est un état particulier qui relie deux régions simultanées. Les régions peuvent être des pairs, c'est-à-dire deux régions simultanées appartenant au même composite. Ces régions peuvent être soit imbriquées dans des régions homologues à différents niveaux, soit ne pas être liées séquentiellement.

Une **transition** relie la sortie d'une **bifurcation** dans une région à l'entrée de la état de synchronisation, et une autre transition relie la sortie de l'état de synchronisation à l'entrée d'une **jointure** dans l'autre région. En d'autres termes, un état de synchronisation est un tampon qui relie indirectement une bifurcation dans une région à une jonction dans une autre région. La bifurcation et

Chaque jointure doit avoir un état d'entrée et un état de sortie au sein de sa propre région.

Le déclenchement d'une transition dans la première région est mémorisé par l'état de synchronisation jusqu'au déclenchement de la transition de jonction dans la seconde région. Si les conditions de la transition de jonction sont satisfaites avant que l'état de synchronisation ne soit actif, la jonction doit attendre que... transition dans la première région incendies. En d'autres termes, cela représente une transition dans le La deuxième région ne peut pas se déclencher tant qu'une transition dans la première région ne s'est pas déclenchée. Sinon, elle se bloque jusqu'au déclenchement de la première transition. Notez que chaque bifurcation et jonction doit avoir une entrée et une sortie dans sa propre région, les états de synchronisation n'en ont pas ne modifient pas non plus le comportement séquentiel fondamental de chaque région concurrente. ils modifient les règles d'imbrication pour la formation d'états composites (sauf qu'un état synchrone et ses arcs n'appartiennent à aucune des régions concurrentes, mais plutôt à leur superétat composite).

Une situation producteur-consommateur est un exemple typique d'état synchronisé. Le producteur déclenche une transition qui active l'état synchronisé ; le consommateur effectue une transition qui nécessite un état de synchronisation avant de pouvoir se déclencher.

Si la transition d'entrée vers l'état de synchronisation fait partie d'une boucle, il est possible que La première région peut prendre l'avantage sur la seconde. Autrement dit, l'état de synchronisation Il pourrait y avoir plus d'un jeton dessus (pour reprendre la terminologie des réseaux de Petri). Par conséquent, un L'état de synchronisation, contrairement à un état normal, représente un compteur ou une file d'attente (cette dernière si des informations circulent entre les régions ; si l'état de synchronisation est un état de flux d'objets, pour (par exemple). Par défaut, un état de synchronisation peut contenir un nombre illimité de jetons, mais Le modélisateur peut spécifier une limite supérieure au nombre d'états de synchronisation qu'un état peut contenir. La capacité de l'état de synchronisation est dépassée, ce qui représente une erreur d'exécution. Le plus souvent, la limite est illimitée ou égale à 1, cette dernière valeur représentant un verrou simple. Une limite de On n'utiliserait généralement la valeur 1 que s'il est possible de garantir qu'aucun dépassement ne se produira. C'est la responsabilité du modélisateur.

S'il existe des jetons dans un état de synchronisation lorsque l'état composite englobant sort, Ils sont détruits. À chaque entrée dans la région composite, l'état de synchronisation est modifié. vide.

Il peut y avoir plus d'un arc d'entrée entrant dans un état de synchronisation, mais tous doivent provenir de bifurcations dans la même région séquentielle. De même, il peut y en avoir davantage. qu'un seul arc de sortie quittant un état de synchronisation pour fusionner dans la région séquentielle. Comme chaque région est séquentielle, il n'y a aucun risque de conflit entre plusieurs arcs.

Un état de synchronisation peut être un état de flux d'objets. Dans ce cas, il représente une file d'attente de des valeurs qui passent d'une région à l'autre.

Notation

L'état de synchronisation est représenté par un petit cercle contenant une limite supérieure, soit un entier, soit une étoile (*) indiquant une valeur illimitée. Une flèche de transition indique la direction de la synchronisation.

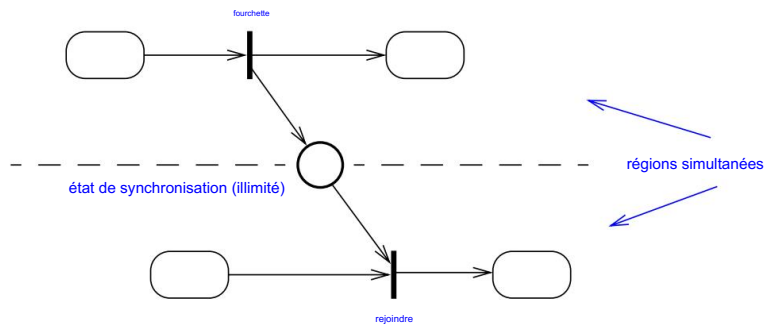


Figure 13-177. Configuration de l'état de synchronisation

un symbole de barre de synchronisation (une barre épaisse) dans l'état de synchronisation, et une autre flèche de transition de l'état de synchronisation vers un symbole de barre de synchronisation dans une autre région (Figure 13-177).

L'état de synchronisation est de préférence dessiné sur la frontière entre deux régions, mais cela n'est pas toujours possible (les régions peuvent ne pas être adjacentes), et la topologie de la connexion est, dans tous les cas, sans ambiguïté.

Dans un diagramme d'activité, chaque arc de transition représente implicitement un état.

Par conséquent, une flèche peut être tracée de la sortie d'une bifurcation à l'entrée d'une jonction sans indiquer explicitement l'état de synchronisation (mais l'état de synchronisation est nécessaire pour indiquer une limite explicite).

L'exemple

de la figure 13-178 illustre un diagramme d'états d'une situation d'achat de billets. L'émission des billets et la facturation sont simultanées, à l'exception de la sélection des sièges qui doit être effectuée avant le calcul et la validation des modifications. Cette synchronisation est représentée par l'insertion d'un état de synchronisation entre « Sélection des sièges » et « Validation des modifications ». Une bifurcation est présente après la sélection des sièges, car elle est suivie de l'impression des billets et de l'état de synchronisation. L'impression des modifications n'est pas soumise à l'attente de la synchronisation. Une jonction est présente avant la validation des modifications, car cette dernière doit attendre la validation du compte et l'état de synchronisation. Une fois l'impression des billets et la validation des modifications terminées, l'état composite se termine et l'envoi des billets par courrier est effectué.

L'état de synchronisation a une limite de un. Une limite supérieure est inutile, car il n'y a qu'une seule synchronisation par exécution de l'état composite.

La figure 13-179 illustre un traitement par lots du processus de traitement des commandes. Dans cette variante, de nombreuses commandes sont traitées hors ligne. Le traitement est effectué par un serveur, tandis que les modifications sont apportées par un autre. Une modification ne peut être effectuée avant le traitement de la commande correspondante, mais le traitement des commandes peut anticiper les frais, ce qui permet une synchronisation illimitée. Il s'agit d'une situation classique de type producteur-consommateur.

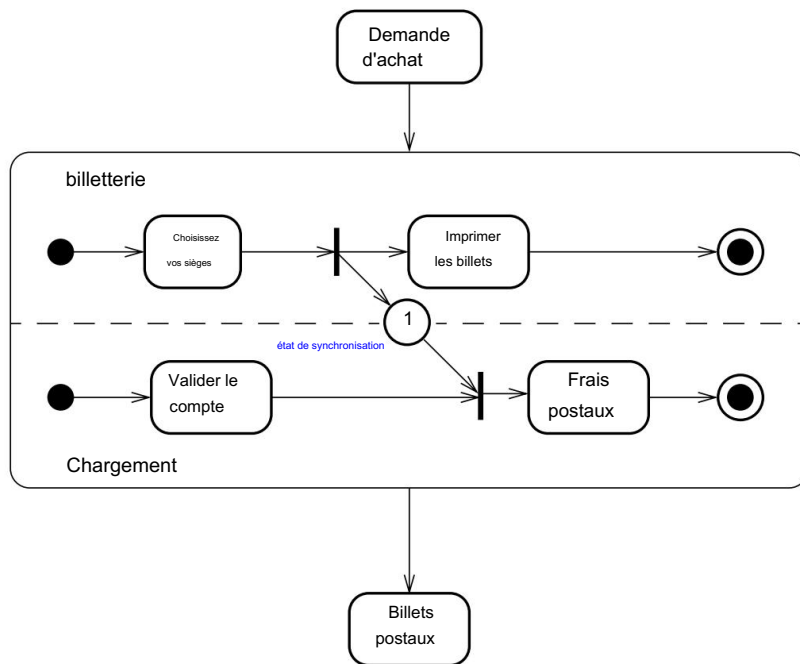


Figure 13-178. État de synchronisation pour un seul ordre

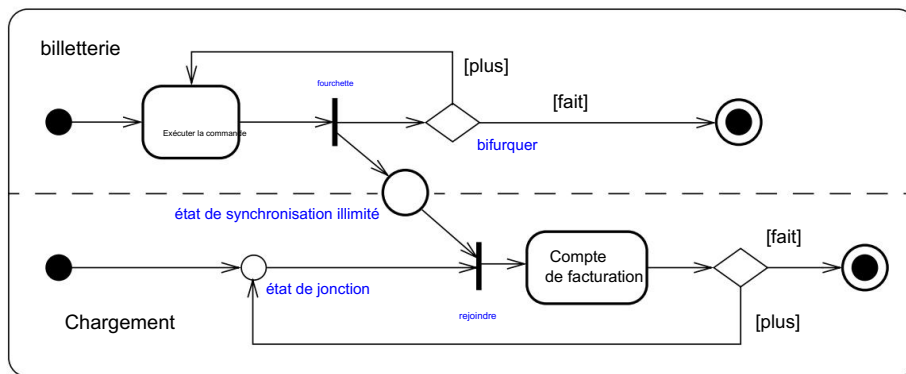


Figure 13-179. Situation producteur-consommateur avec état de synchronisation illimité

Discussion

Les états synchrones permettent de modéliser des situations producteur-consommateur avec une surcharge minimale et une sécurité accrue par rapport aux constructions de concurrence plus générales, car chaque région concurrente maintient un seul fil de contrôle à tout moment.

Il n'y a aucun risque de dépassement (si la limite de l'état de synchronisation est illimitée), car le L'état de synchronisation est vidé chaque fois que le superétat concurrent est quitté. Si les états de synchronisation sont utilisées dans des boucles contenant des branches, mais il existe un risque de blocage. Dans ce cas, une région a terminé sa synchronisation, mais l'autre attend un jeton de synchronisation qui n'arrivera jamais. Un blocage est également possible si chaque région se trouve dans une branche. qui attend un jeton provenant de l'autre région. Il n'existe aucun moyen d'éviter complètement ce problème. des situations dans les systèmes concurrents qui permettent de prendre des décisions. Même sans états de synchronisation, L'arrêt complet ne peut être garanti en raison du problème d'interruption.

action synchrone

Une requête durant laquelle l'objet émetteur attend une réponse ; un **appel**. À l'inverse : **une action asynchrone**.

système

Un ensemble d'unités interconnectées, organisées pour atteindre un objectif. Un système peut être décrit par un ou plusieurs modèles, éventuellement selon différents points de vue. Le système est le « modèle complet ».

Sémantique

Le système est modélisé par un **sous**-système de niveau supérieur qui contient indirectement l'ensemble ensemble d'éléments de modèle qui fonctionnent ensemble pour réaliser un monde réel complet but.

Notation

Un système peut être représenté comme un ensemble, sous le stéréotype du « système ». Il est cependant rarement nécessaire de le représenter comme une unité.

étiqueter

La valeur du sélecteur dans une paire **valeur-balise** . Elle représente le nom d'une propriété définie lors de la modélisation.

valeur étiquetée

Une paire étiquette-valeur associée à un élément pour contenir une information.

Voir aussi **contrainte**, **stéréotype**.

Voir **le chapitre 14, Éléments standard**, pour une liste des balises prédéfinies.

Sémantique

Une valeur étiquetée est une paire sélecteur-valeur qui peut être associée à n'importe quel élément (y compris les éléments de modèle et les éléments de présentation) pour véhiculer divers types d'informations, généralement secondaires par rapport à la sémantique de l'élément, mais potentiellement importantes pour celle-ci. Modélisation d'entreprise. Le sélecteur est appelé une **étiquette** ; il s'agit d'une **chaîne de caractères**. Chaque **étiquette** représente une propriété qui peut s'appliquer à un seul type d'élément ou à plusieurs.

Dans un modèle, le nom d'une balise ne peut apparaître qu'une seule fois sur chaque élément. Sa valeur peut être différente de celle de l'élément précédent.

Il peut s'agir de différents types, mais la valeur est encodée sous forme de **chaîne de caractères**. L'interprétation de la valeur est la suivante : une convention entre le modélisateur et l'outil de modélisation. On s'attend à ce que

Les valeurs étiquetées seront implémentées sous forme de tables de consultation indexées par étiquettes pour une acquisition efficace. **taxe**

Les valeurs étiquetées représentent des informations arbitraires exprimées sous forme de texte et sont couramment utilisés pour stocker des informations de gestion de projet, telles que l'auteur d'un l'élément, l'état des tests ou l'importance de l'élément pour un système final (le Les balises peuvent être l'auteur, le statut et l'importance).

Les valeurs étiquetées constituent une extension modeste des méta-attributs d'UML métaclasses. Il ne s'agit pas d'un mécanisme d'extension entièrement général, mais il peut être utilisé pour ajouter des informations aux classes **de métamodèles** existantes pour le bénéfice des outils back-end, tels que les générateurs de code, les outils de création de rapports et les simulateurs. Pour éviter toute confusion, des balises sont utilisées. devraient différer des méta-attributs existants des éléments de modèle auxquels ils sont associés. appliqué. Ce contrôle peut être facilité par un outil de modélisation.

Certaines étiquettes sont prédéfinies dans UML ; d'autres peuvent être définies par l'utilisateur. Les valeurs étiquetées constituent un mécanisme d'extensibilité permettant d'associer des informations arbitraires aux modèles.

Notation

Chaque valeur étiquetée est affichée sous la forme

étiquette = **valeur**

où **tag** est le nom d'une étiquette et **value** est une valeur littérale. Les valeurs étiquetées peuvent être inclus avec d'autres mots-clés relatifs à la propriété dans une **liste de propriétés** séparées par des virgules et encadrées par des accolades.

Un mot-clé peut être déclaré pour représenter une étiquette ayant une valeur particulière. Dans ce cas, Le mot-clé peut être utilisé seul. L'absence de la balise est considérée comme équivalente à une des autres valeurs légales de l'étiquette.

étiqueter

Exemple

```
{ auteur=Joe, statut=testé, exigence=3.563.2a, supprimer }
```

Discussion

La plupart des logiciels d'édition de modèles offrent des fonctionnalités de base pour la définition, l'affichage et la modélisation. Ils recherchent les valeurs étiquetées comme des chaînes de caractères, mais ne les utilisent pas pour étendre la sémantique UML. Cependant, les outils back-end, tels que les générateurs de code, les générateurs de rapports et les Par exemple, il est possible de lire les valeurs étiquetées afin de modifier leur sémantique de manière flexible. Notez que Les listes de valeurs étiquetées sont une idée ancienne, par exemple les listes de propriétés dans le langage Lisp.

Les valeurs étiquetées sont un moyen d'associer une gestion de projet non sémantique et Informations de suivi pour les modèles. Par exemple, l'étiquette « auteur » peut contenir l'auteur de L'état d'un élément et d'une balise peut contenir l'état de développement, par exemple incomplet.

Testé, bogué et complet.

Les valeurs étiquetées permettent également d'associer des éléments dépendants du langage d'implémentation. des contrôles vers un modèle UML sans intégrer les détails du langage dans UML.

Les indicateurs, les indications et les pragmas du générateur de code peuvent être encodés sous forme de valeurs étiquetées sans affectant le modèle sous-jacent. Plusieurs ensembles d'étiquettes sont possibles pour différentes langues sur un même modèle. Ni un éditeur de modèle ni un analyseur sémantique n'en ont besoin.

Il faut comprendre les balises, car elles peuvent être manipulées comme des chaînes de caractères. Un outil back-end, tel que... un générateur de code peut comprendre et traiter les valeurs étiquetées. Par exemple, un

La valeur étiquetée peut nommer la classe de conteneur utilisée pour remplacer l'implémentation par défaut d'une association à multiplicité « many ».

Les valeurs étiquetées répondent au besoin de nombreux types d'informations qui doivent peuvent être associés à des modèles, mais ils ne sont pas conçus comme une extensibilité complète du métamodèle mécanisme. Les étiquettes forment un espace de noms plat qui doit être géré en adoptant des conventions afin d'éviter les conflits de noms. Elles ne prévoient pas la possibilité de spécifier

Les types de valeurs qu'elles contiennent ne sont pas précisés. Elles ne sont pas non plus destinées à des extensions sémantiques importantes du langage de modélisation lui-même. Les étiquettes sont quelque peu similaires aux attributs du métamodèle, mais elles n'en sont pas et n'ont pas été formalisées en tant qu'attributs du métamodèle. tel.

L'utilisation de balises, comme l'utilisation de procédures dans les bibliothèques de langages de programmation, Cela peut nécessiter une période d'évolution durant laquelle des conflits peuvent survenir entre les développeurs. Avec le temps, certaines normes d'utilisation se développeront. UML n'inclut pas de

Il ne s'agit pas d'un « registre » d'étiquettes, ni de la possibilité pour les premiers utilisateurs de les « réutiliser » afin d'empêcher d'autres utilisations ultérieures.

lunette de visée

Une spécification indiquant si une valeur est une instance ou un classificateur.

Voir **la portée**.

Discussion

La portée cible sert principalement à stocker des classes comme valeurs d'attributs ou cibles d'association. Son utilité est limitée. Le terme « **portée** » désigne à lui seul **la portée propriétaire**.

état cible

L' état **de la machine à états** résultant du déclenchement de la **transition**. Après qu'un objet a géré un événement provoquant le déclenchement d'une transition , l'objet se trouve dans l'état cible. de la transition (ou des états cibles s'il s'agit d'une **transition complexe** avec plusieurs cibles) (états). Non applicable à une **transition interne**, qui n'entraîne pas de changement de État.

Voir **la transition**.

modèle

Un **élément de modèle paramétré**. Pour l'utiliser, les **paramètres** doivent être liés (à (temps du modèle) aux valeurs réelles. Synonyme : **élément paramétré**.

Voir aussi **liaison, élément lié**.

Sémantique

Un modèle est le descripteur d'un élément comportant un ou plusieurs paramètres formels non liés. Il définit donc une famille d'éléments potentiels, chaque élément étant spécifié. en associant les paramètres à des valeurs réelles. Généralement, les paramètres sont des classificateurs. qui représentent des types d'attributs, mais ils peuvent également représenter des entiers ou même des opérations. Les éléments subordonnés au sein du modèle sont définis en fonction de la formalité paramètres, donc eux aussi sont liés lorsque le modèle lui-même est lié à des paramètres réels valeurs.

Une classe modèle est le descripteur d'une classe paramétrée. Le corps d'un modèle peut contenir des occurrences des paramètres formels, ainsi qu'un élément par défaut représentant le modèle lui-même. Une classe réelle est produite par la liaison des éléments.

Les paramètres peuvent être associés à des valeurs. Les attributs et les opérations au sein de la classe modèle peuvent être défini en fonction des paramètres formels. La classe modèle peut également avoir des relations, telles que des associations et des généralisations, entre elle-même et l'un de ses paramètres. Lorsque le modèle est lié, le résultat est une relation entre les classe modèle liée et classes liées aux paramètres de relation.

Une classe modèle n'est pas directement utilisable car elle possède des paramètres non liés. Ses paramètres doivent être liés à des valeurs concrètes pour créer une classe réelle. Seule une classe réelle Une classe peut être le parent ou la cible d'une association (une association unidirectionnelle de Le modèle vers une autre classe est toutefois autorisé. Une classe modèle peut être une sous-classe d'une classe ordinaire, ce qui implique que toutes les classes formées en liant les Les modèles sont des sous-classes de la classe donnée. Ils peuvent également être des enfants de l'un des paramètres de modèle ; cela implique que la classe de modèle liée est un enfant de la classe. passé comme argument.

La paramétrisation peut être appliquée à d'autres éléments du modèle, tels que les collaborations et même des packages entiers. La description donnée ici pour les classes s'applique à d'autres éléments de modélisation de manière évidente.

Le contenu d'un modèle n'est pas directement soumis aux règles de bonne formation des modèles. C'est parce qu'ils incluent des paramètres dont la sémantique n'est pas complète. jusqu'à ce qu'ils soient liés. Un modèle est une sorte d'élément de modèle de second niveau, et non un qui modélise directement les systèmes, mais qui modélise d'autres éléments du modèle. Le contenu d'un modèle est donc extérieur à la sémantique du système. Les résultats de Les éléments de modèle ordinaires qui lient un gabarit sont soumis à des règles de bonne formation et constituent des éléments normaux du système cible. Certaines règles de bonne formation Les règles relatives aux modèles pourraient être déduites du principe que leurs résultats liés doivent être bien formés, mais nous n'essaierons pas de les énumérer. En un sens, lorsqu'un Le modèle est lié, son contenu est dupliqué et les paramètres sont remplacés par les arguments. Le résultat s'intègre au modèle effectif comme s'il y avait été inclus directement.

D'autres types de classificateurs, tels que **les cas d'utilisation** et **les signaux**, peuvent être paramétrés. Les collaborations peuvent également être paramétrées ; elles deviennent alors **des modèles**.

Notation

Un petit rectangle en pointillés est superposé au coin supérieur droit de la classe rectangle ou autre élément de modélisation. Le rectangle en pointillés contient une liste de paramètres formels pour la classe. Chaque paramètre possède un nom et un classificateur. Le champ ne doit pas être vide (sinon, il n'y a pas de modèle), même s'il peut être omis lors de la présentation. Le nom, les attributs et les opérations de la classe paramétrée apparaissent normalement dans le rectangle de classe, mais les occurrences du champ formel Des paramètres peuvent également être inclus. D'autres types d'éléments paramétrés existent. Ils sont traités de manière similaire. Les paramètres formels peuvent apparaître à l'intérieur du corps du modèle, par exemple pour indiquer une classe apparentée identifiée par l'un des paramètres.

Les paramètres ont la syntaxe suivante

nom : **type**

où **nom** est un identifiant pour le paramètre, avec une portée à l'intérieur du modèle ; et

où **type** est une chaîne de caractères désignant une **expression de type** pour le paramètre.

Si le nom du type est omis, il est considéré comme une expression de type qui se résout en un Un classificateur, tel qu'un nom de classe ou un type de données, doit être spécifié. Les autres types de paramètres (comme Integer) doivent être indiqués explicitement et correspondre à des expressions de type valides.

La figure 13-180 présente un modèle avec un paramètre entier et un paramètre de classe. Ce modèle est associé à l'un de ses paramètres.

Discussion

Le modèle effectif est le modèle implicite résultant de la liaison de tous les modèles ; il est Le modèle implicite qui décrit un système. Les paramètres du modèle n'ont aucune signification. au sein même du modèle effectif, car ils auront été liés. Ils peuvent être

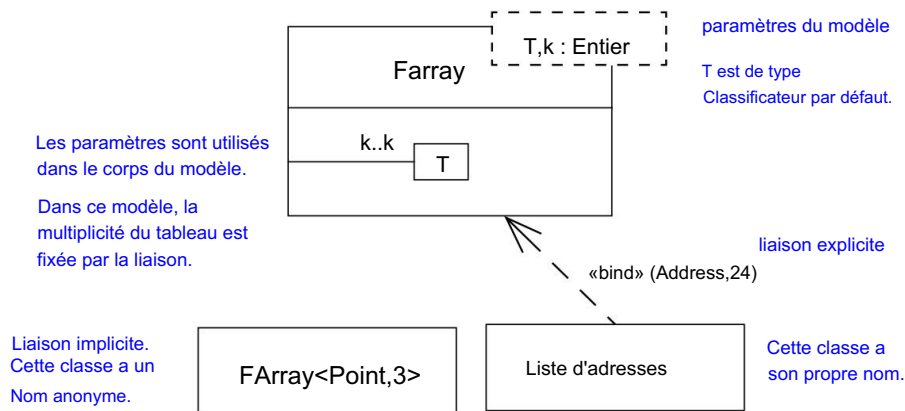


Figure 13-180. Notation de modèle avec utilisation d'un paramètre comme référence

Utilisé uniquement dans le cadre du corps du modèle lui-même. Cela suffit pour gérer les éléments constitutifs contenus dans l'élément paramétré, par exemple, pour les attributs ou les opérations d'une classe paramétrée.

La gestion des éléments généralement externes à l'élément paramétré se complexifie. Par exemple, une classe peut avoir des associations ou des généralisations vers d'autres classes. Si ces classes sont des paramètres du modèle, elles ne peuvent pas faire partie du modèle effectif, mais elles ne font pas non plus partie d'une classe ordinaire. C'est pourquoi un élément paramétré comprend un corps représentant un fragment de modèle. Ce fragment ne fait pas partie du modèle effectif. Il fait partie du modèle lui-même et peut inclure des paramètres de modèle, comme un paramètre représentant une classe (encore non spécifiée). Lors de la liaison du modèle, le corps est implicitement copié, les paramètres sont remplacés par des arguments et la copie devient une partie du modèle effectif. Chaque instantiation du modèle ajoute un élément au modèle effectif. La figure 13-181 en donne un exemple.

Le corps du modèle contient implicitement un élément représentant l'élément de modèle instancié lui-même — par exemple, la classe produite par la liaison d'un modèle. Cet élément implicite permet de construire des relations, telles que des associations et des généralisations, avec les paramètres du modèle. Dans la notation, les paramètres sont représentés à l'intérieur des limites du modèle, et une connexion à l'intérieur de ces limites indique une relation avec l'élément de modèle implicitement instancié. Lors de l'instanciation du modèle, ces relations se transforment, dans le modèle effectif, en relations entre l'élément de modèle lié (nouvellement instancié) et les éléments (précédemment existants) qui sont des arguments du modèle.

Un modèle peut être un enfant d'un autre élément. Cela signifie que chaque élément lié généré à partir de celui-ci est un enfant de l'élément donné. Par exemple, dans la figure 13-181, chaque tableau de variables (VArray) est un tableau (Array). Par conséquent, VArray<Point> est un enfant de Array, VArray<Address> est un enfant de Array, et ainsi de suite.

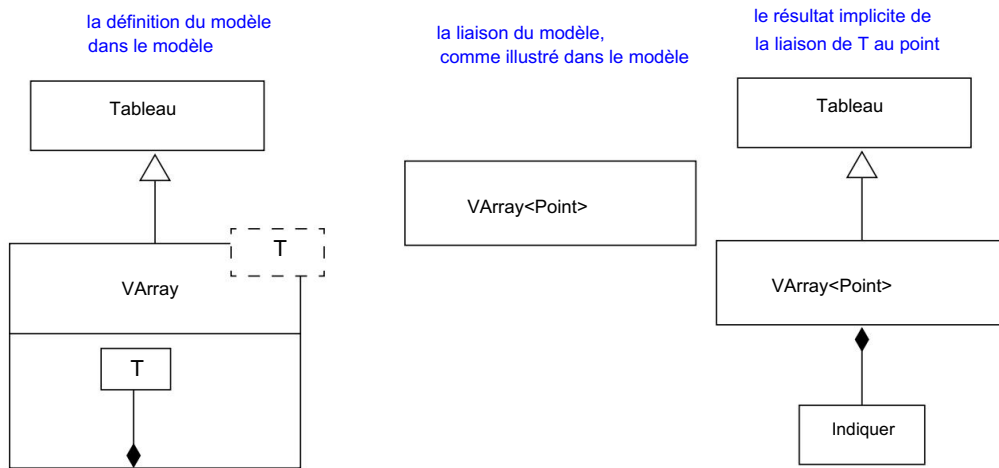


Figure 13-181. Modèle avec relation avec l'un de ses paramètres

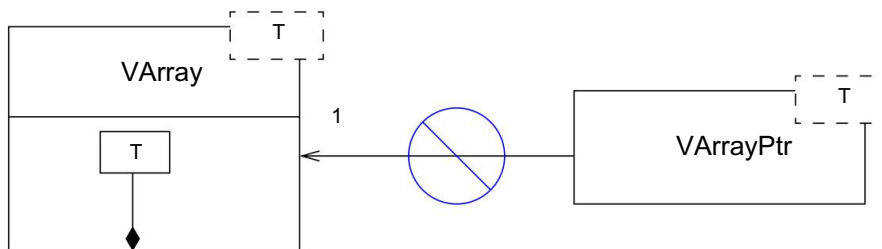
Un modèle ne peut généralement pas être le parent d'un autre élément. Cela signifierait que chaque élément généré par la liaison du modèle serait le parent de l'autre élément. Bien qu'il soit possible d'expliquer une telle situation, elle est peu plausible.

Deux modèles ne sont pas associés simplement parce qu'ils partagent le même nom de paramètre. (Tenter de les associer reviendrait à lier chaque instanciation du premier modèle à chaque instanciation du second, ce qui n'est généralement pas souhaité. Ce point a souvent été mal compris par les auteurs par le passé.) Un paramètre n'a de portée qu'à l'intérieur de son propre modèle. Utiliser le même nom pour un paramètre dans deux modèles ne signifie pas qu'il s'agit du même paramètre.

En général, si deux modèles comportent des éléments paramétrés qui doivent être liés, l'un des modèles doit être instancié à l'intérieur du corps de l'autre. (Rappelons qu'un modèle est implicitement instancié à l'intérieur de son propre corps. Par conséquent, les deux modèles sont effectivement instanciés à l'intérieur du corps, et les relations s'établissent donc entre les éléments instanciés.) La figure 13-182 illustre une tentative incorrecte et une tentative correcte de définition d'une telle relation — ici, avec une classe « pointeur » paramétrée qui pointe vers un tableau paramétré du même type.

Une approche similaire permet de déclarer une classe paramétrée, enfant d'une autre classe modèle liée par le même paramètre. Une autre approche consiste à instancier les deux modèles dans un troisième modèle possédant un paramètre. Ce paramètre sert à lier une copie de chacun des autres modèles. Une association peut alors être établie entre les copies instanciées. Dans la plupart des cas pratiques, cela n'est pas nécessaire, car la relation peut être déclarée dans le corps de l'un des modèles.

FAUX ! Cela n'a aucun sens. Les T n'ont aucun lien entre eux car ils appartiennent à des contextes différents.



Exactement ! Un modèle doit être instancié pour construire une association.

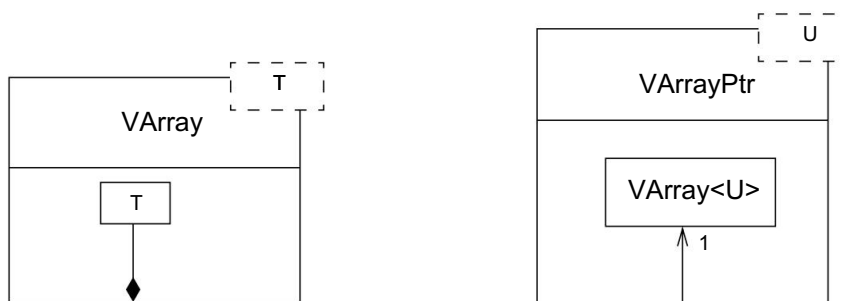


Figure 13-182. Associations entre les modèles

fil

(Issu de l'expression « fil de contrôle ») Un chemin d'exécution unique au sein d'un programme, d'un modèle dynamique ou d'une autre représentation du flux de contrôle. Également un stéréotype désignant l'implémentation d'un objet actif sous forme de processus léger.

Voir **objet actif**, **transition complexe**, **état composite**, **machine à états**, **synchronisation** **État**.

temps

Une valeur représentant un instant absolu ou relatif dans le temps.

Voir l'expression **temporelle**.

événement temporel

Un **événement** qui marque la réalisation d'une **expression temporelle**, comme la survenue d'un temps absolu ou le passage d'un laps de temps donné après l'entrée d'un objet.
un **État**.

Sémantique

Un événement temporel est un événement qui dépend du passage du temps et donc de l'existence d'une horloge. Dans le monde réel, l'horloge est implicite. Dans un ordinateur, c'est une entité physique, et il peut y avoir différentes horloges d'un ordinateur à l'autre. L'événement temporel est un message envoyé par l'horloge au système. Il est important de noter que le temps absolu et le temps écoulé peuvent être définis par rapport à une horloge réelle ou à une horloge interne virtuelle (dans ce dernier cas, celle-ci peut varier d'un objet à l'autre).

Les événements temporels peuvent être basés sur le temps absolu (l'heure du jour ou le réglage d'une horloge au sein d'un système) ou sur le temps relatif (le temps écoulé depuis l'entrée dans un certain état ou la survenue d'un événement).

Notation

Les événements temporels ne sont pas déclarés comme des événements nommés, contrairement aux signaux. Une **expression temporelle** sert simplement de déclencheur à une transition.

Discussion

Dans toute implémentation concrète, les événements temporels ne proviennent pas de l'univers ; ils sont générés par une horloge interne ou externe au système. De ce fait, ils deviennent presque indiscernables des signaux, notamment dans les systèmes temps réel et distribués. Dans de tels systèmes, il est également nécessaire de déterminer quelle horloge est utilisée ; le « temps réel » n'existe pas en soi. (Il n'existe d'ailleurs pas dans l'univers réel ; il suffit de demander à Einstein.)

expression temporelle

Une **expression** qui se résout en une valeur de temps absolue ou relative. Utilisée pour définir un **événement temporel**.

Sémantique

La plupart des expressions temporelles correspondent soit au temps écoulé depuis l'entrée dans un état, soit à la survenue d'un instant précis. D'autres expressions temporelles doivent être définies de manière ad hoc.

Notation

Temps écoulé. Un événement désignant le passage d'une certaine durée après l'entrée dans l'état contenant la transition est noté avec le mot-clé « après » suivi d'une expression qui s'évalue (au moment de la modélisation) à une durée.

après (10 secondes)

après (10 secondes depuis la sortie de l'état A)

Si aucun point de départ n'est spécifié, il s'agit du temps écoulé depuis l'entrée dans l'état contenant la transition.

Temps absolu. Un événement désignant la survenue d'un temps absolu est noté avec le mot-clé « when », suivi d'une expression booléenne entre parenthèses impliquant le temps.

quand (date = 1er janvier 2000)

marque de chronométrage

Une indication temporelle signalant l'occurrence d'un **événement** ou d' **un message** . Les repères temporels peuvent être utilisés dans **les contraintes**.

Sémantique

Une marque de synchronisation est créée à partir du nom d'un message. Lors d'une interaction, un message peut être nommé afin de pouvoir créer des expressions de marque de synchronisation à partir de ce nom. Dans les expressions suivantes, « **message** » désigne le nom d'un message.

message.sendTime()	Heure d'envoi du message
message.receiveTime()	Heure à laquelle ce message est reçu

Notation

Les expressions de repères temporels sont affichées sous forme de texte.

Exemple :

La contrainte suivante limite le temps nécessaire pour produire une tonalité.

{ dialtone.sendTime () – offhook.sendTime () < 1 seconde }

tracer

Une **dépendance** qui indique un processus de développement historique ou une autre relation extra-modèle entre deux éléments représentant le même concept, sans règles spécifiques permettant de dériver l'un de l'autre. C'est le type de dépendance le moins spécifique, et sa sémantique est minimale. Elle sert surtout de rappel pour la pensée humaine en développement.

Voir **dépendance**, **modèle**.

Sémantique

Une trace est une forme de **dépendance** qui indique un lien entre deux éléments représentant le même concept à différents niveaux de signification. Elle ne représente pas la sémantique au sein d'un **modèle**, mais plutôt les liens entre des éléments ayant des sémantiques différentes, c'est-à-dire entre des éléments appartenant à des modèles différents.

sur différents plans de signification. Il n'y a pas de correspondance explicite entre les éléments. Souvent, cela représente une connexion entre deux manières de saisir un concept.

à différents stades de développement. Par exemple, deux éléments qui sont des variations

Des éléments portant sur le même thème peuvent être liés par une trace. Une trace ne représente pas une relation entre des instances d'exécution, mais plutôt une dépendance entre les éléments du modèle eux-mêmes.

L'une des principales utilisations du traçage est le suivi des exigences qui ont été modifiées.

tout au long du développement d'un système, les dépendances de traçage peuvent relier des éléments appartenant à deux types de modèles (comme un modèle de cas d'utilisation et un modèle de conception) ou à des modèles de traçage. deux versions du même type de modèle.

Notation

Une trace est indiquée par une flèche de dépendance (une flèche en pointillés dont la queue est sur le côté).

(élément plus récent et sa tête sur l'élément plus ancien) avec le mot-clé « trace ». Cependant, généralement, les éléments appartiennent à des modèles différents qui ne sont pas affichés simultanément ; en pratique, la relation est donc le plus souvent implémentée dans un outil.

sous forme de lien hypertexte.

lien transitoire

Un **lien** qui existe pour une durée limitée, par exemple pour l'exécution d'une opération.

Voir aussi **association, collaboration, usage**.

Sémantique

Lors de l'exécution, certains **liens** n'existent que pour une durée limitée. Bien sûr, presque tous

Un objet ou un lien a une durée de vie limitée, si la période est suffisamment longue. Certains liens,

toutefois, elles n'existent que dans certains contextes limités, comme lors de l'exécution d'une

méthode. Les arguments de procédure et les variables locales peuvent être représentés par des variables transitoires.

liens. Il est possible de modéliser tous ces liens comme **des associations**, mais alors les conditions

Les associations doivent être formulées de manière très générale, et elles perdent une grande partie de leur précision

pour contraindre les combinaisons d'objets. De telles situations peuvent en revanche être modélisées à l'aide **de**

collaborations, qui sont des configurations d'objets et de liens existant.

dans des contextes particuliers.

Le **rôle d'association** au sein d'une collaboration peut être considéré comme un lien transitoire qui

Elle n'existe que lors de l'exécution d'une entité comportementale, telle qu'une procédure. Elle apparaît dans un modèle

de classe comme une **dépendance d'utilisation**. Pour plus de détails, il est nécessaire de

consulter le modèle comportemental.

Notation

Un lien transitoire est représenté comme une **association** avec un **stéréotype** attaché au lien.
Ce rôle permet d'indiquer différents types de mise en œuvre. Les stéréotypes suivants peuvent être utilisé :

«paramètre»	Paramètre de procédure
«local»	Variable locale d'une procédure
"mondial"	Variable globale (élément visible dans l'ensemble du modèle) ou emballage); à éviter, si possible, car cela contrevient à l'esprit de orientation objet
"soi"	Auto-liaison (la capacité d'un objet à envoyer un message à elle-même, implicite dans les objets et utile à montrer seulement dans situations dynamiques avec flux de messages)
"association"	Association (par défaut, inutile de spécifier sauf pour (soulignement) ; il ne s'agit pas d'un lien temporaire, mais il est mentionné par souci d'exhaustivité.

objet transitoire

Un objet qui n'existe que pendant l'exécution du **thread** qui l'a créé.

transition (phase)

La quatrième phase d'un **processus de développement logiciel**, durant laquelle le système implémenté est configuré pour son exécution dans un contexte réel.
phase, la vue de déploiement est terminée, ainsi que toutes les autres vues qui n'avaient pas été réalisées lors des phases précédentes.
Voir **le processus de développement**.

transition

Une relation au sein d'une **machine à états** entre deux **états** indiquant qu'un objet Dans le premier état, effectuera **des actions** spécifiques et passera au deuxième état lorsqu'un **L'événement** spécifié se produit et **les conditions de garde** spécifiées sont satisfaites. Dans un tel cas, Lors d'un changement d'état, on dit que la transition se **déclenche**. Une **transition simple** possède une seule source. Un état source et un seul état cible. Une **transition complexe** possède plusieurs états sources. et/ou plusieurs états cibles. Cela représente une modification du nombre d'états actifs simultanément, ou une bifurcation ou une jonction de contrôle. Une **transition interne** a une source état mais pas d'état cible. Il représente une réponse à un événement sans changement d'état. L'état. Les états et les transitions sont les sommets et les nœuds des machines à états.
Voir aussi **machine à états**.

Sémantique

Les transitions représentent les chemins potentiels entre les **états** de l'histoire de vie d'un objet, ainsi que les actions effectuées lors d'un changement d'état. Une transition indique la manière dont un objet, dans un état donné, réagit à la survenue d'un événement. Les états et les transitions sont les sommets et les arcs d'une **machine à états** qui décrit les histoires de vie possibles des instances d'un classificateur.

Structure

Une transition possède un état source, un déclencheur d'événement, une condition de garde, une action et un état cible. Certains de ces éléments peuvent être absents lors d'une transition.

État source. L' **état source** est l'état affecté par la transition. Si un objet se trouve dans l'état source, une transition sortante de cet état peut **être déclenchée** si l'objet reçoit l' événement **déclencheur** de la transition et si la **condition de garde** (le cas échéant) est satisfaite.

État cible. L' **état cible** est l'état actif après la fin de la transition. Il s'agit de l'état vers lequel l'objet maître évolue. L'état cible n'est pas utilisé lors d'une **transition interne**, qui n'entraîne aucun changement d'état.

Déclencheur d'événement. Le **déclencheur** d'événement est l' **événement** dont la réception par l'objet dans l' **état source** rend la transition éligible à son exécution, à condition que sa **condition de garde** soit satisfaite. Si l'événement possède **des paramètres**, leurs valeurs sont accessibles à la transition et peuvent être utilisées dans **les expressions** de la condition de garde et **des actions**. L'événement déclenchant une transition devient l' **événement courant** et est accessible par les actions suivantes faisant partie de l' étape **d'exécution jusqu'à son terme**, initiée par cet événement.

Une transition sans événement déclencheur explicite est appelée **transition d'achèvement** (ou **transition sans déclencheur**) et se déclenche implicitement à la fin de toute **activité** interne dans l'état. Un état composite indique son achèvement en atteignant son **état final**. Si un état ne contient aucune activité interne ni états imbriqués, une transition d'achèvement se déclenche immédiatement lors de l'entrée dans cet état après l'exécution de toute **action d'entrée** . Notez qu'une transition d'achèvement doit satisfaire sa condition de garde pour se déclencher. Si la condition de garde est fausse lors de l'achèvement, l'événement d'achèvement implicite est consommé et la transition ne se déclenchera pas ultérieurement, même si la condition de garde devient vraie. (Ce type de comportement peut être modélisé à l'aide d'un **événement de changement**.)

Notez que toutes les occurrences d'un événement au sein d'une machine à états doivent avoir la même signature.

Condition de garde. La **condition de garde** est une **expression booléenne** évaluée lorsqu'une transition est **déclenchée** par la gestion d'un événement, y compris un événement de fin implicite lors d'une **transition de fin**. Si la machine à états exécute une étape de traitement jusqu'à son achèvement lorsqu'un événement survient, celui-ci est enregistré jusqu'à la fin de l'étape et l'arrêt de la machine à états. Sinon, l'événement est traité.

immédiatement. Si l'expression est évaluée à vrai, la transition peut être **déclenchée**. Si

Si l'expression est évaluée à faux, la transition n'est pas déclenchée. Si aucune transition n'est déclenchée,

Si l'événement devient éligible au déclenchement, il est ignoré. Il ne s'agit pas d'une erreur. Plusieurs transitions ayant des conditions de garde différentes peuvent être déclenchées par le même événement.

L'événement se produit, toutes les conditions de garde sont testées. Si plusieurs conditions de garde sont testées, elles sont vérifiées.

En effet, une seule transition sera déclenchée. Le choix de la transition à déclencher est indéterminé si aucune règle de priorité n'est définie.

Notez que la condition de garde n'est évaluée qu'une seule fois, au moment où l'événement se produit. est gérée. Si la condition est initialement fausse puis devient vraie, la transition ne se déclenchera pas à moins qu'un autre événement ne survienne et que la condition soit vraie à ce moment-là.

Notez qu'une condition de garde n'est pas la méthode appropriée pour surveiller en continu un valeur. Un **événement de changement** devrait être utilisé dans une telle situation.

Si une transition n'a pas de condition de garde, alors la condition de garde est traitée comme La transition est activée si son événement déclencheur se produit. Si plusieurs transitions Si plusieurs options sont activées, une seule sera déclenchée. Le choix peut être aléatoire.

Par commodité, une condition de garde peut être décomposée en une série de conditions de garde plus simples. conditions. En fait, plusieurs conditions de garde peuvent découler d'un seul déclencheur. événement ou condition de garde. Chaque chemin dans l'arbre représente une transition unique. déclenché par l'événement déclencheur (unique) avec une condition de garde effective différente Il s'agit de la conjonction (« et ») des conditions de garde le long de son chemin. Toutes les expressions le long de ce chemin sont évaluées avant qu'une transition ne soit choisie pour être déclenchée. Une transition ne peut pas se déclencher partiellement. En effet, un ensemble de transitions indépendantes peut partager une partie de leur description. La figure 13-183 en donne un exemple.

Notez que les arbres de conditions de garde et la possibilité d'ordonner les transitions pour l'éligibilité ne sont que des commodités, car le même effet pourrait être obtenu par un ensemble de transitions indépendantes, chacune avec sa propre condition de garde disjointe.

Action. Une transition peut contenir une **expression d'action** qui décrit une **action**.

Il s'agit d'une expression pour un calcul procédural qui peut affecter l'objet qui possède la machine à états (et, indirectement, d'autres objets auxquels elle peut accéder). Cette expression peut utiliser **les paramètres** de l'événement **déclencheur**, ainsi que **les attributs** et **associations** de l'objet propriétaire. L'événement déclencheur est disponible en tant **qu'événement courant**. pendant toute la **durée de l'étape d'exécution jusqu'à son terme**, initiée par l'événement, y compris les étapes ultérieures. segments sans déclencheur et actions d'entrée et de sortie.

Une action peut être une **séquence d'actions**. Une action est atomique — c'est-à-dire qu'elle ne peut pas être atomique. L'exécution doit être interrompue de l'extérieur et doit être entièrement terminée avant que toute autre action ou tout autre événement puisse être traité. Les actions doivent être de durée minimale afin d'éviter Suspendre la machine à états. Tous les événements reçus lors de l'exécution d'une action sont Elles sont conservées jusqu'à la fin de l'action, moment auquel elles sont évaluées.

Branches. Par commodité, plusieurs transitions qui partagent le même événement déclencheur mais les différentes conditions de garde peuvent être regroupées dans le modèle et la notation afin d'éviter la duplication de la gâchette ou de la partie commune de la condition de garde.

Voir **la branche** pour plus de détails sur la représentation et la notation.

L'état (l' état cible) est désigné par une chaîne de transition. La figure 13-184 illustre une transition entre deux états et la division de l'un en segments.

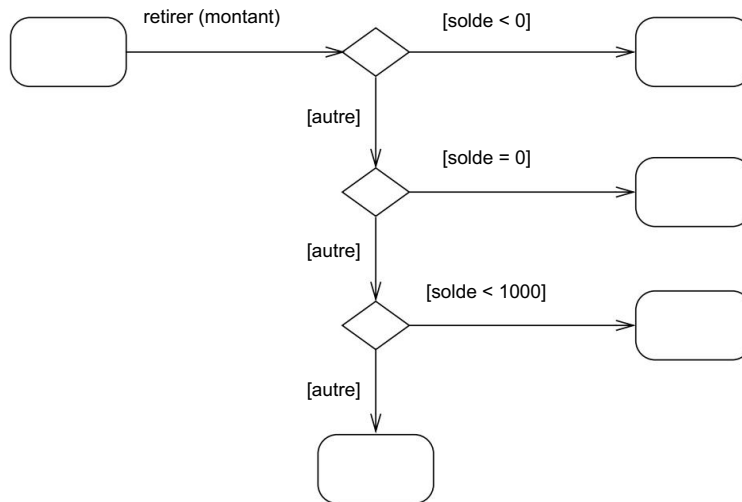


Figure 13-183. Arbre des conditions de garde

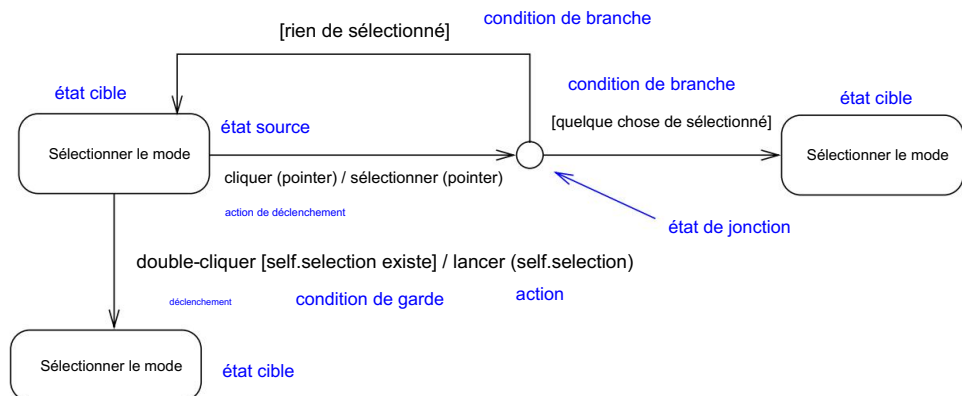


Figure 13-184. Transitions

Une **transition interne** est représentée par une chaîne de transition à l'intérieur d'un symbole d'état. Une chaîne de transition a le format suivant :

nom :opt nom-événementopt (liste-paramètres)opt [condition-garde]opt / liste-actionopt

Ce nom peut être utilisé pour faire référence à la transition dans les expressions, notamment pour former **les repères de chronométrage**. Il est suivi de deux points.

Le nom de l'événement est suivi de ses paramètres. La liste des paramètres peut être omise s'il n'y a aucun paramètre. Le nom de l'événement et la liste des paramètres sont omis lors d'une **transition de fin**. La **liste des paramètres** a le format suivant :

Nom :typelist. La

condition de garde est une expression booléenne formulée en fonction des paramètres de l'événement déclencheur, ainsi que des attributs et liens de l'objet décrit par la machine à états. Cette condition peut également inclure des tests sur l'état d'états concurrents de la machine courante ou sur des états explicitement désignés d'un objet accessible (par exemple : [dans l'état 1] et [pas dans l'état 2]) . Les noms d'état peuvent être entièrement qualifiés par les états imbriqués qui les contiennent, ce qui donne des chemins d'accès de la forme État1::État2::État3. Ceci est utile si un même nom d'état apparaît dans différentes régions d'états composites de la machine globale.

La liste d'actions est une expression procédurale exécutée lorsque la transition est déclenchée. Elle peut être formulée en termes d'opérations, d'attributs et de liens de l'objet propriétaire, ainsi que des paramètres de l'événement déclencheur. Les actions peuvent inclure des appels, des envois et d'autres types d'actions. La **liste d'actions** peut contenir plusieurs clauses d'action séparées par des points-virgules : **liste_actions** ;

Branches. Une transition peut comprendre un segment avec un événement déclencheur suivi d'un arbre d' **états de jonction**, représentés par de petits cercles. Ceci équivaut à un ensemble de transitions individuelles, une pour chaque chemin dans l'arbre, dont la condition de garde est le « et » de toutes les conditions le long du chemin. Seul le segment final de chaque chemin peut avoir une action.

Un état de jonction peut être représenté indifféremment par un losange, qu'il s'agisse d'une branche ou d'une fusion. Cela n'a aucune incidence sur sa signification.

Discussion

Une transition représente un changement atomique d'un état à un autre, éventuellement accompagné d'une action atomique. Une transition est irréversible. Les actions sur une transition doivent être des calculs courts, généralement triviaux, tels que des affectations et des opérations arithmétiques simples.

temps de transition

Voir [le repère de synchronisation](#).

déclenchement

Un [événement](#) dont la survenue rend une [transition](#) susceptible d'être déclenchée. Le mot peut être utilisé comme nom (pour l'événement lui-même) ou comme verbe (pour l'occurrence de l'événement).

Voir aussi [transition d'achèvement](#), [transition](#).

Sémantique

Chaque transition (à l'exception d'une [transition de fin](#) qui se déclenche à la fin d'une activité interne) contient une référence à un [événement](#) dans sa structure. Si l'événement se produit lorsqu'un objet se trouve dans un [état](#) contenant une transition sortante dont le déclencheur est

l'événement ou un ancêtre de l'événement, alors la [condition de garde](#) sur la transition est testé. Si la condition est remplie, la transition est activée. Si la

Si la condition de garde est absente, elle est considérée comme satisfaite. Si plusieurs transitions sont éligibles, une seule sera déclenchée. Le choix peut être indéterministe. (Si l'objet possède plusieurs états simultanés, une transition de...)

Chaque état peut être déclenché. Mais au plus une transition par état peut être déclenchée.

Notez que l'état de la garde est testé une seule fois, au moment du déclenchement.

Un événement se produit (y compris un événement de fin implicite). Si aucune transition n'est activée Pour déclencher une alarme suite à un événement, celui-ci est simplement ignoré. Il ne s'agit pas d'une erreur. ror.

Les [paramètres](#) de l'événement déclencheur sont disponibles pour être utilisés dans une condition de garde ou une [action](#) associée à la transition ou à une [action d'entrée](#) sur l'état cible.

Tout au long de l'exécution d'une étape de finalisation après une transition, le L'événement déclencheur reste disponible pour les actions des sous-étapes de la transition en tant que l' [événement actuel](#). Le type exact de cet événement peut être inconnu dans une action de saisie ou dans un segment ultérieur d'une transition à plusieurs segments. Par conséquent, le type d'événement Il est possible de distinguer les événements dans une action à l'aide d'une opération polymorphe ou d'une instruction conditionnelle. Une fois le type d'événement exact connu, ses paramètres peuvent être utilisés.

Notation

Le nom et la signature de l'événement déclencheur font partie de l'étiquette d'une transition.

Voir [la transition](#).

L'événement déclencheur peut être accédé dans une expression par le mot réservé `currentEvent`. Ce mot-clé fait référence à l'événement déclencheur du premier segment d'un Transition à segments multiples.

transition sans déclencheur

Une transition sans déclencheur d'événement explicite. Lorsqu'elle quitte un état normal, elle représente une **transition de fin**, c'est-à-dire une transition déclenchée par un événement.

l'achèvement d'une activité plutôt que par un événement explicite. Lorsqu'il quitte un **pseudo-état**, il représente un segment de transition qui est automatiquement parcouru lorsque le

Le segment précédent a terminé son action. Les transitions sans déclencheur sont utilisées pour relier **les états initiaux** et **les états historiques** à leurs états cibles.

tuple

Une liste ordonnée de valeurs. Généralement, le terme implique l'existence d'un ensemble de telles listes. de forme similaire. (Il s'agit d'un terme mathématique standard.)

taper

En tant qu'adjectif : Le **classificateur** déclaré qui qualifie la valeur d'un attribut, d'un paramètre ou d'un La variable doit contenir une valeur. Cette valeur doit être une **instance** du type ou l'une de ses instances. **descendance**.

En tant que nom : Un **stéréotype** de **classe** utilisé pour spécifier un ensemble d' **instances** (telles que **des objets**). ainsi que les **opérations** applicables aux objets. Un type ne peut contenir aucun **méthode**. Contraste : **interface**, **classe d'implémentation**.

classe de type et d'implémentation

Les classes peuvent être stéréotypées comme **des types** ou **des classes d'implémentation** (bien qu'elles puissent (Il peut également rester indifférencié). Un type sert à spécifier un domaine d' **objets**, ainsi que **les opérations** applicables à ces objets, sans définir l'implémentation physique des objets ou des opérations. Un type ne peut pas inclure **de méthodes**, mais il

peut fournir des spécifications comportementales pour ses opérations. Il peut également inclure **des attributs** et **des associations** afin de spécifier le comportement des opérations.

Les attributs et les associations ne déterminent pas l'implémentation de ses objets.

Une **classe d'implémentation** définit la structure de données physique et les méthodes des objets, telles qu'implémentées dans les langages traditionnels comme C++ et Smalltalk. On dit qu'une classe d'implémentation réalise **un** type si elle inclut toutes les méthodes nécessaires.

les opérations sont du même type et ont le même comportement. Une classe d'implémentation peut réaliser plusieurs types, et plusieurs classes d'implémentation peuvent réaliser le même type. Les attributs et associations d'une classe d'implémentation n'ont pas besoin d'être identiques à celles d'un type qu'elle réalise. La classe d'implémentation peut fournir méthodes de fonctionnement en termes d'attributs et d'associations physiques, et peut déclarer des opérations supplémentaires qui ne figurent dans aucun type.

Un objet doit être une instance d'au plus une classe d'implémentation, qui spécifie l'implémentation physique de l'objet. Cependant, un objet peut être un in-

position de plusieurs types. Si l'objet est une instance d'une classe d'implémentation, La classe d'implémentation doit alors identifier les types dont l'objet est une instance. Si **une classification dynamique** est disponible, un objet peut acquérir ou perdre des types.

au cours de sa durée de vie. Un type utilisé de cette manière caractérise un rôle changeant qu'un objet peut adopter puis abandonner.

Bien que l'utilisation des types et des classes d'implémentation diffère, leur fonctionnement interne reste le même. La structure étant identique, elles sont modélisées comme des stéréotypes de classe. Elles sont pleinement prises en charge. **la généralisation** ; le **principe de substituabilité** ; et l' **héritage** des attributs, des associations et des opérations. Les types ne peuvent toutefois spécialiser que des types, et les classes d'implémentation ne peuvent spécialiser que des classes d'implémentation. Les types peuvent être lié aux classes d'implémentation uniquement par **réalisation**.

Notation

Une classe non différenciée est représentée sans mot-clé. Un type est représenté par le mot-clé « type », et une classe d'implémentation par « implementation class ».

La figure 13-185 en montre un exemple.

L'implémentation d'un type par une classe d'implémentation est modélisée à l'aide de relation **de réalisation**, représentée par une ligne pointillée avec une pointe de flèche triangulaire pleine (une « flèche **de généralisation** en pointillés » ou une « **dépendance** avec pointe de flèche pleine »). Un symbole implique l'**héritage** des **opérations**, mais pas de la structure (**attributs** ou **associations**). La réalisation peut être utilisée entre deux classificateurs quelconques dont l'un prend en charge les opérations de l'autre, mais l'implémentation d'un type par un

La classe d'implémentation est une utilisation courante de cette technologie.

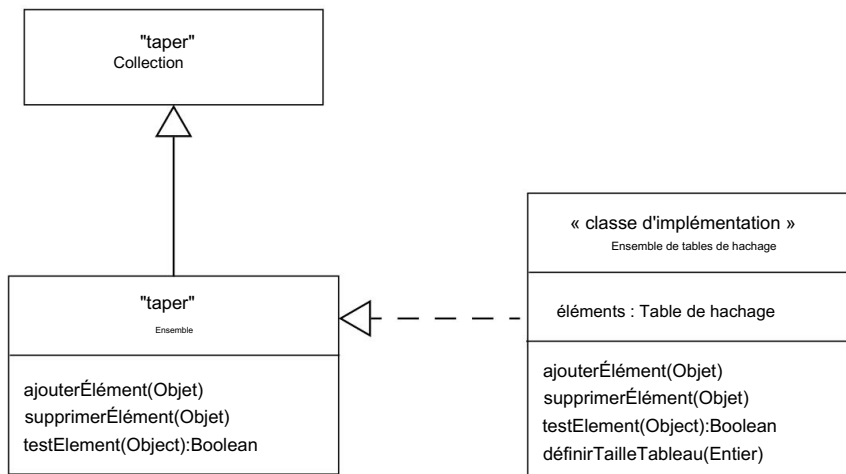


Figure 13-185. Notation des types et des classes d'implémentation

Discussion

Les concepts de type et de classe d'implémentation sont relativement limités et destinés aux langages de programmation conventionnels, tels que C++ et Smalltalk. Le concept de classe UML peut être utilisé directement et de manière plus générale. UML prend en charge la **classification multiple** et la **classification dynamique**, ce qui rend superflue la distinction entre types et classes d'implémentation. Cette distinction reste néanmoins utile pour la conception de code dans les langages conventionnels.

expression de type

Une **expression** qui s'évalue en une **référence** à un ou plusieurs **types de données**. Par exemple, une déclaration de type d'attribut dans un langage de programmation classique est une expression de type.

Exemple

En C++, voici quelques expressions de type :

Personne*

Ordre[24]

Booléen (*) (Chaîne de caractères, entier)

non interprété

Un espace réservé pour un ou plusieurs types dont l'implémentation n'est pas spécifiée par UML. Chaque valeur non interprétée possède une représentation sous forme de chaîne de caractères. Dans toute implémentation physique, comme un éditeur de modèles, l'implémentation doit être complète ; il n'y a donc pas de valeurs non interprétées.

valeur non spécifiée

Une valeur qui n'a pas encore été spécifiée par le modélisateur.

Discussion

Une valeur non spécifiée n'est pas une valeur au sens strict et ne peut figurer dans un modèle complet, sauf pour les propriétés dont la valeur est inutile ou non pertinente. Par exemple, la multiplicité ne peut être inconnue ; elle doit avoir une valeur. L'absence de toute connaissance équivaut à une multiplicité de plusieurs. La sémantique d'UML n'autorise donc pas et ne traite pas l'absence de valeurs ou les valeurs non spécifiées.

Il existe un autre sens dans lequel « non spécifié » constitue une partie utile d'un modèle inachevé. Cela signifie : « Je n'ai pas encore réfléchi à cette valeur et je l'ai notée afin de me souvenir de lui attribuer une valeur plus tard. » C'est une formulation explicite.

affirmation selon laquelle le modèle est incomplet. Il s'agit d'une fonctionnalité utile et qui Des outils peuvent être utiles. Par définition, une telle valeur ne peut apparaître dans un produit fini. Il s'agit d'un modèle, et il est inutile d'en définir la sémantique. Un outil peut automatiquement fournir une valeur par défaut pour une valeur non spécifiée lorsqu'une valeur est nécessaire ; par exemple, Lors de la génération de code, les valeurs non spécifiées sont utilisées par commodité, sans incidence sur la sémantique. Elles sont en dehors du champ sémantique d'UML.

De même, la valeur par défaut d'une propriété n'a aucune signification sémantique. Une propriété du modèle possède simplement une valeur. Un outil peut fournir automatiquement des valeurs pour les autres propriétés. propriétés des éléments nouvellement créés. Mais encore une fois, il s'agit simplement d'une commodité opérationnelle au sein de l'outil, et non d'un élément de la sémantique UML. UML sémantiquement complet Les modèles n'ont pas de valeurs par défaut ou non spécifiées ; ils ont simplement des valeurs.

usage

Une **dépendance** dans laquelle un élément (le client) requiert la présence d'un autre. élément (le fournisseur) pour son bon fonctionnement ou sa mise en œuvre — généralement, pour des raisons de mise en œuvre.

Voir aussi **collaboration**, **dépendance**, **lien transitoire**.

Sémantique

Une dépendance **d'utilisation** est une situation dans laquelle un élément nécessite la présence de un autre élément pour sa mise en œuvre ou son fonctionnement correct. Tous les éléments doivent exister au même niveau de signification, c'est-à-dire qu'elles n'impliquent pas de changement de niveau de signification. niveau d'abstraction ou de réalisation (tel qu'une correspondance entre un niveau d'analyse) classe et classe de niveau implémentation). Fréquemment, une dépendance d'utilisation implique éléments au niveau de l'implémentation, tels qu'un fichier d'inclusion C++, pour lesquels cela implique Conséquences du compilateur. Un usage peut être stéréotypé davantage pour indiquer l'usage exact. la nature de la dépendance, comme l'appel d'une opération d'une autre classe ou l'instanciation d'un objet d'une autre classe.

Notation

Une utilisation est indiquée par une flèche en pointillés (dépendance) avec le mot-clé « utilisation ». La pointe de la flèche se trouve du côté du fournisseur (indépendant), et la pointe du trait du côté du client. élément (dépendant).

Discussion

Un usage correspond généralement à un lien **transitoire**, c'est-à-dire à une connexion entre des instances de classes qui n'est pas significative ou présente en permanence, mais seulement dans certains cas. Dans ce contexte, par exemple lors de l'exécution d'une procédure de sous-routine, la construction de dépendance ne modélise pas l'intégralité des informations, mais seulement le fait qu'elle existe.

existence. Le modèle **de collaboration** offre la possibilité de modéliser ces relations dans leurs moindres détails.

Éléments standard

appeler, créer, instancier, envoyer

utiliser

Mot-clé pour la dépendance **d'utilisation** dans la notation.

cas d'utilisation

La spécification des séquences d'actions, y compris les séquences variantes et les erreurs séquences qu'un système, un sous-système ou une classe peut exécuter en interagissant avec des acteurs externes.

Voir aussi **acteur, classificateur**.

Sémantique

Un cas d'utilisation est une unité de fonctionnalité cohérente fournie par un **classificateur** (un **système**, **sous-système**, ou **classe**) tel que manifesté par des séquences de messages échangés entre les le système et un ou plusieurs utilisateurs externes (représentés comme **acteurs**), ainsi que les actions effectuées par le système.

L'objectif d'un cas d'utilisation est de définir un comportement d'un classificateur (y compris un sous-système ou le système entier), sans révéler sa structure interne.

le classificateur. Chaque cas d'utilisation spécifie un service que le classificateur fournit à ses utilisateurs.

Autrement dit, il s'agit d'une manière spécifique d'utiliser le classificateur, visible de l'extérieur. Elle décrit une séquence complète initiée par un utilisateur (modélisée par un **acteur**) en termes

de l'interaction entre les utilisateurs et le classificateur, ainsi que des réponses effectuées

par le classificateur. L'interaction comprend uniquement les communications entre les

Le système et les acteurs. Le comportement interne ou l'implémentation est masqué. L'ensemble des cas d'utilisation d'un classificateur ou d'un système partitionne et couvre son comportement.

Le cas d'utilisation représente une fonctionnalité quantifiée significative disponible pour

utilisateurs. Notez que le terme « utilisateur » inclut les humains, ainsi que les ordinateurs et autres objets.

L'acteur est une idéalisation de l'objectif d'un utilisateur, et non une représentation physique.

Un utilisateur physique peut correspondre à plusieurs acteurs, et un acteur peut représenter l'utilisateur.

même aspect de plusieurs utilisateurs physiques.

Voir **l'acteur**.

Un cas d'utilisation inclut le comportement normal du système en réponse à une requête utilisateur, comme ainsi que les variantes possibles de la séquence normale, telles que les séquences alternatives, les comportements exceptionnels et la gestion des erreurs. L'objectif est de décrire un élément cohérent fonctionnalités dans toutes leurs variantes, y compris toutes les conditions d'erreur. L'intégralité

Un ensemble de cas d'utilisation pour un classificateur spécifie toutes les différentes manières d'utiliser ce classificateur. Pour plus de commodité, les cas d'utilisation peuvent être regroupés en packages.

Un cas d'utilisation est un **descripteur** ; il décrit un comportement potentiel. L'exécution d'un cas d'utilisation est une **instance de cas d'utilisation**. Le comportement d'un cas d'utilisation peut être spécifié par une **machine à états** associée ou par du code texte (qui est équivalent à une machine à états). peut également être décrit par une description textuelle informelle. Le comportement peut être illustré, mais pas formellement spécifié, par un ensemble de scénarios. Mais aux premiers stades du développement, Cela peut suffire.

Une instance de cas d'utilisation est une exécution d'un cas d'utilisation, initiée par un message provenant d'un instance d'un acteur. En réponse au message, l'instance du cas d'utilisation exécute une séquence d'actions spécifiée par le cas d'utilisation, comme l'envoi de messages aux instances d'acteurs, et pas nécessairement uniquement à l'acteur initiateur. Les instances d'acteurs peuvent envoyer des messages à l'instance de cas d'utilisation, et l'interaction se poursuit jusqu'à ce que l'instance Elle a répondu à toutes les entrées. Lorsqu'elle n'attend plus d'entrées, elle s'arrête.

Un cas d'utilisation est une spécification du comportement d'un système (ou d'un autre classificateur) en tant que L'ensemble des interactions avec les acteurs externes est décrit par une collaboration qui concrétise un cas d'utilisation. Les interactions internes entre les objets internes d'un système qui met en œuvre le comportement sont décrites par une **collaboration** qui réalise un cas d'utilisation.

Structure

Un cas d'utilisation peut comporter des caractéristiques et des relations de classification.

Caractéristiques . Un cas d'utilisation est un **classificateur** et possède donc **des attributs et des opérations**.

Les attributs sont utilisés pour représenter l'état du cas d'utilisation, c'est-à-dire la progression de son exécution. Une opération représente une tâche que le cas d'utilisation peut effectuer. non directement appelable de l'extérieur, mais peut être utilisé pour décrire l'effet de cas d'utilisation sur le système. L'exécution d'une opération peut être associée à réception d'un message d'un acteur. Les opérations agissent sur les attributs de l'utilisateur et indirectement sur le système ou la classe auxquels le cas d'utilisation est rattaché.

Associations aux acteurs. Une **association** entre un acteur et un cas d'utilisation indique

L'instance d'acteur communique avec l'instance du système ou l'instance du classificateur pour produire un résultat qui l'intéresse. Les acteurs modélisent les utilisateurs externes d'un classificateur. Ainsi, si le classificateur est un système, ses acteurs sont les utilisateurs externes. du système. Les acteurs des sous-systèmes de niveau inférieur peuvent appartenir à d'autres classes au sein du système global.

Un acteur peut communiquer avec plusieurs cas d'utilisation — c'est-à-dire que l'acteur peut demander plusieurs services différents du système — et un cas d'utilisation peut communiquer avec un ou plusieurs acteurs lors de la fourniture de son service. Notez deux cas d'utilisation qui précisent que le même système ne peut pas communiquer entre eux car chacun des Chacune d'elles décrit individuellement une utilisation complète du système. Elles peuvent interagir indirectement par le biais d'acteurs partagés.

L'interaction entre les acteurs et les cas d'utilisation peut être définie par des interfaces. Une interface définit les opérations qu'un acteur ou un cas d'utilisation peut prendre en charge ou utiliser. Différentes interfaces proposées par un même cas d'utilisation ne sont pas nécessairement disjointes.

Les cas d'utilisation sont liés à d'autres cas d'utilisation par **généralisation**, **extension** et **inclusion** de relations.

Généralisation. Une relation **de généralisation** relie un cas d'utilisation spécialisé à un cas d'utilisation plus général. Le cas d'utilisation enfant hérite des attributs, opérations et séquences de comportement du cas d'utilisation parent et peut ajouter ses propres attributs et opérations. Le cas d'utilisation enfant enrichit le comportement du cas d'utilisation parent en insérant des séquences d'actions supplémentaires à des points arbitraires. Il peut également modifier certaines opérations et séquences héritées, mais cette modification doit être effectuée avec la même précaution que toute autre redéfinition afin de préserver l'intention du cas d'utilisation parent.

Toute relation d'inclusion ou d'extension avec le cas d'utilisation enfant modifie également le comportement hérité du cas d'utilisation parent.

Extension. Une relation **d'extension** est un type de **dépendance**. Le cas d'utilisation client ajoute des fonctionnalités au cas d'utilisation de base en y insérant des séquences d'actions supplémentaires. Le cas d'utilisation client contient un ou plusieurs segments de séquence de comportement distincts. La relation d'extension contient une liste de noms **de points d'extension** du cas d'utilisation de base, en nombre égal au nombre de segments du cas d'utilisation client. Un point d'extension représente un ou plusieurs emplacements dans le cas d'utilisation de base où l'extension peut être insérée. Une relation d'extension peut également comporter une condition, qui peut utiliser des attributs du cas d'utilisation parent.

Lorsqu'une instance du cas d'utilisation parent atteint un emplacement référencé par un point d'extension dans une relation d'extension, la condition est évaluée ; si elle est vraie, le segment de comportement correspondant du cas d'utilisation enfant est exécuté. En l'absence de condition, celle-ci est considérée comme toujours vraie. Si la relation d'extension comporte plusieurs points d'extension, la condition est évaluée uniquement au niveau du premier point d'extension avant l'exécution du premier segment.

Une relation d'extension ne crée pas un nouveau cas d'utilisation instanciable. Elle ajoute implicitement un comportement au cas d'utilisation de base initial. Ce dernier inclut implicitement le comportement étendu. Le cas d'utilisation de base initial non étendu n'est plus disponible sous sa forme originale. Autrement dit, si vous étendez un cas d'utilisation, vous ne pouvez pas instancier explicitement le cas d'utilisation de base sans la possibilité d'extensions. Un cas d'utilisation peut avoir plusieurs extensions qui s'appliquent toutes au même cas d'utilisation de base et peuvent être insérées dans une seule instance de cas d'utilisation, si leurs conditions respectives sont remplies.

En revanche, un cas d'utilisation d'extension peut étendre plusieurs cas d'utilisation de base (ou le même cas d'utilisation à différents points d'extension), chacun à son propre point d'extension (ou liste de points d'extension). Si plusieurs extensions sont présentes au même point d'extension, leur ordre d'exécution relatif est non déterministe.

Notez que le cas d'utilisation d'extension ne doit pas être instancié ; le cas d'utilisation de base doit être instancié pour obtenir le comportement combiné de base et d'extensions.

Le cas d'utilisation de sion peut être instanciable ou non, mais dans tous les cas, il n'inclut pas le comportement du cas d'utilisation de base.

Inclure. Une relation **d'inclusion** désigne l'inclusion de la séquence de comportements de le cas d'utilisation du fournisseur dans la séquence d'interaction d'un cas d'utilisation du client, sous la contrôle du cas d'utilisation du client à un emplacement spécifié par le client dans sa description.

Il s'agit d'une dépendance, et non d'une généralisation, car le cas d'utilisation du fournisseur ne peut être substitué là où apparaît le cas d'utilisation du client. Le client peut accéder aux attributs de la base pour obtenir des valeurs et communiquer des résultats.

L'instance exécute le cas d'utilisation client. Lorsqu'elle atteint le point d'inclusion, elle commence à exécuter le cas d'utilisation fournisseur jusqu'à ce que celui-ci soit terminé. Ensuite, elle reprend l'exécution du cas d'utilisation client. le cas d'utilisation du client au-delà du lieu d'inclusion. Les attributs de l'utilisation du fournisseur Les valeurs de cas ne persistent pas d'une exécution à l'autre.

Un cas d'utilisation peut être **abstrait**, ce qui signifie qu'il ne peut pas être instancié directement dans l'exécution d'un système. Il définit un fragment de comportement spécialisé par ou inclus dans des cas d'utilisation concrets, ou il peut s'agir d'une extension d'un cas d'utilisation de base. Il peut Il doit également être **concret** s'il peut être instancié par lui-même.

Comportement. La séquence de comportement d'un cas d'utilisation peut être décrite à l'aide d'une machine à états finis, d'un graphe d'activités ou de code texte dans un langage exécutable. Les actions de La machine à états ou les instructions du code peuvent faire appel aux opérations internes de Le cas d'utilisation permet de préciser les effets de l'exécution. Les actions peuvent également indiquer l'envoi de messages aux acteurs.

Un cas d'utilisation peut être décrit de manière informelle à l'aide de scénarios ou de texte brut, mais tel Les descriptions sont imprécises et destinées uniquement à l'interprétation humaine.

Les actions d'un cas d'utilisation peuvent être spécifiées en termes d'appels aux opérations de la classificateur que décrit le cas d'utilisation. Une opération peut être appelée par plusieurs un cas d'utilisation.

Réalisation. La **réalisation** d'un cas d'utilisation peut être spécifiée par un ensemble de collaborations. Une **collaboration** décrit l'implémentation du cas d'utilisation par des objets.

le classificateur décrit par le cas d'utilisation. Chaque collaboration décrit le contexte

parmi les constituants du système au sein duquel se produisent une ou plusieurs séquences **d'interaction**. Les collaborations et leurs interactions définissent comment les objets au sein du système interagissent.

Le système interagit pour atteindre le comportement externe spécifié dans le cas d'utilisation.

Un système peut être spécifié par des cas d'utilisation à différents niveaux d'abstraction.

Un cas spécifiant un système, par exemple, peut être affiné en un ensemble d'utilisations subordonnées cas, chacun spécifiant un service d'un sous-système. La fonctionnalité spécifiée par le

Le cas d'utilisation de niveau supérieur est entièrement traçable jusqu'à la fonctionnalité de

Les cas d'utilisation subordonnés (de niveau inférieur). Un cas d'utilisation supérieur et un ensemble de cas d'utilisation subordonnés spécifient le même comportement à deux niveaux d'abstraction. Les cas d'utilisation subordonnés coopèrent pour fournir le comportement du cas d'utilisation supérieur.

La coopération des cas d'utilisation subordonnés est spécifiée par des collaborations de

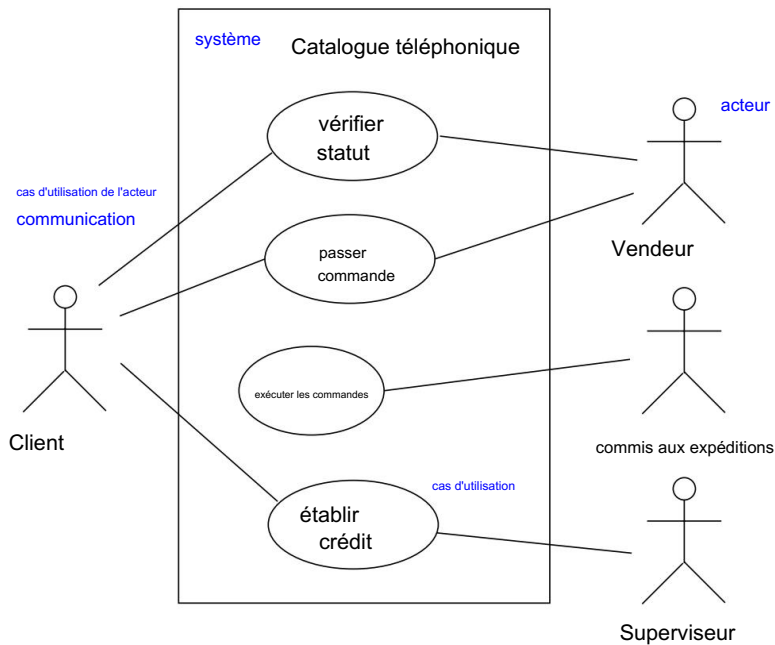


Figure 13-186. Cas d'utilisation et acteurs

Le cas d'utilisation principal peut être représenté dans des diagrammes de collaboration. Les acteurs d'un cas d'utilisation principal apparaissent comme acteurs des cas d'utilisation subordonnés.

De plus, les cas d'utilisation subordonnés interagissent entre eux. Cette réalisation par couches aboutit à un ensemble imbriqué de cas d'utilisation et de collaborations qui mettent en œuvre l'ensemble du système.

Notation

Un cas d'utilisation est représenté par une ellipse contenant son nom. Si les attributs ou les opérations du cas d'utilisation doivent être représentés ; le cas d'utilisation peut être dessiné sous la forme d'un rectangle classificateur avec le mot-clé « cas d'utilisation ». La figure 13-186 montre un diagramme de cas d'utilisation.

Un point d'extension est une entité nommée au sein d'un cas d'utilisation qui décrit des emplacements à partir duquel des séquences d'actions provenant d'autres cas d'utilisation peuvent être insérées. Il fournit un niveau d'indirection entre les extensions et le texte de la séquence de comportement. Un point d'extension fait référence à un emplacement ou à un ensemble d'emplacements au sein de la séquence de comportement de le cas d'utilisation. La référence peut être modifiée indépendamment des relations étendues. qui utilisent le point d'extension. Chaque point d'extension doit avoir un nom unique. dans un cas d'utilisation. Les points d'extension peuvent être listés dans un compartiment du cas d'utilisation. avec les points d'extension de l'en-tête (Figure 13-187).

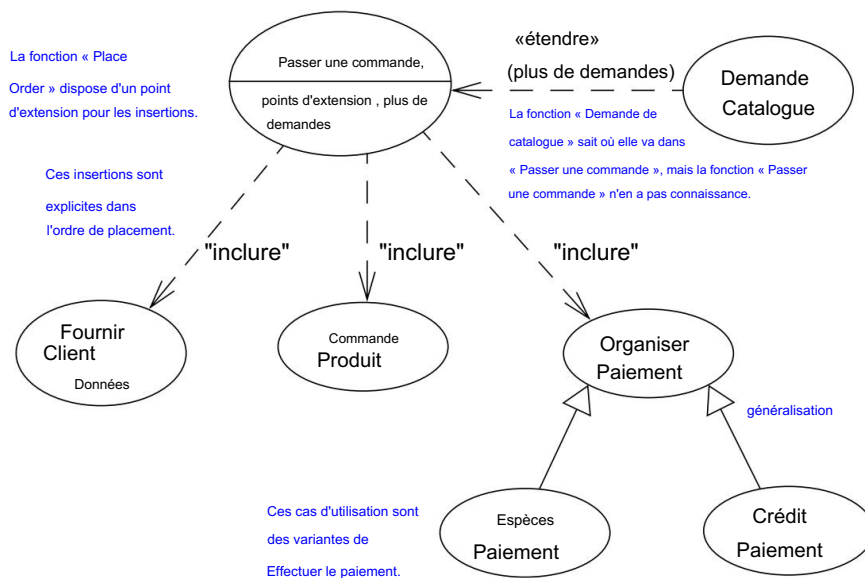


Figure 13-187. Relations entre les cas d'utilisation

Une relation de communication entre un cas d'utilisation et un acteur est représentée par un symbole d'association : un chemin continu entre les symboles du cas d'utilisation et de l'acteur.

Le mot-clé « communication » peut généralement être omis, car il s'agit du seul type d'association entre les acteurs et les cas d'utilisation. En règle générale, aucun nom ni rôle n'est indiqué, car l'acteur et le cas d'utilisation définissent la relation de manière univoque.

Une relation de généralisation est représentée par une flèche de généralisation : un chemin continu allant du cas d'utilisation enfant au cas d'utilisation parent, avec une pointe de flèche triangulaire fermée sur le cas d'utilisation parent.

Une relation d'extension ou d'inclusion est représentée par une flèche de dépendance avec le mot-clé « extend » ou « include » — une ligne pointillée avec une flèche pointillée sur le cas d'utilisation client. Une relation d'extension comporte également une liste de noms de points d'extension (ceux-ci peuvent être masqués dans le diagramme).

La figure 13-187 illustre différents types de relations entre cas d'utilisation.

Spécification du comportement. La relation entre un cas d'utilisation et ses séquences d'interaction externes est généralement représentée par un lien hypertexte vers des diagrammes de séquence. Ce lien est invisible, mais il est possible de le parcourir dans un éditeur. Le comportement peut également être spécifié par une machine à états finis ou par du texte en langage de programmation associé au cas d'utilisation. Un texte en langage naturel peut servir de spécification informelle.

Voir la section « extend » pour un exemple de séquences comportementales.

La relation entre un cas d'utilisation et son implémentation peut être représentée comme une relation **de réalisation** d'un cas d'utilisation à une collaboration. Mais comme il s'agit de

Souvent dans des modèles distincts, il est généralement représenté par un lien hypertexte invisible. On s'attend à ce qu'un outil permette de « zoomer » sur un cas d'utilisation pour en voir les détails.

scénarios et/ou mise en œuvre en collaboration.

diagramme de cas d'utilisation

Un diagramme qui illustre les relations entre les acteurs et les cas d'utilisation au sein d'un système.

Voir **acteur, cas d'utilisation**.

Notation

Un diagramme de cas d'utilisation est un graphique d'acteurs, un ensemble de cas d'utilisation délimités par un système.

Les diagrammes de cas d'utilisation présentent les éléments suivants : limites (un rectangle), associations entre les acteurs et les cas d'utilisation, relations entre les cas d'utilisation et généralisation entre les acteurs.

afficher les éléments du **modèle de cas d'utilisation** (cas d'utilisation, acteurs).

généralisation des cas d'utilisation

Une relation taxonomique entre un **cas d'utilisation** (l' **enfant**) et le cas d'utilisation (le **parent**) qui décrit les caractéristiques que l'enfant partage avec d'autres cas d'utilisation.

ont le même parent. Il s'agit d' **une généralisation** applicable aux cas d'utilisation.

Sémantique

Un cas d'utilisation parent peut être spécialisé en un ou plusieurs cas d'utilisation enfants qui représentent des formes plus spécifiques du parent (**Figure 13-188**). Un enfant hérite de tous les attributs, opérations et relations de son parent, car un cas d'utilisation est un **classificateur**. L'implémentation d'une opération héritée peut être redéfinie par un collaboration qui concrétise un cas d'utilisation pour enfant.

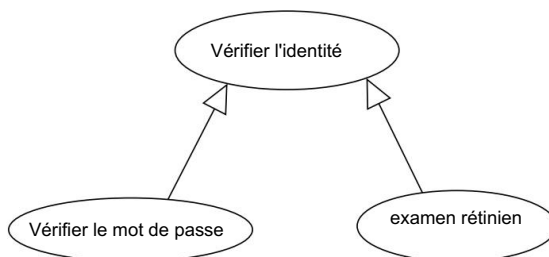


Figure 13-188. Généralisation des cas d'utilisation

Comportement du cas d'utilisation pour la vérification d'identité du parent :

Le parent est abstrait, il n'y a pas de séquence de comportement.

Un descendant concret doit fournir le comportement indiqué ci-dessous.

Comportement du cas d'utilisation pour la vérification du mot de passe enfant :

Récupérer le mot de passe à partir de la base de données principale

Demander le mot de passe

L'utilisateur fournit le mot de passe

Vérifier le mot de passe par rapport à l'entrée de l'utilisateur

Comportement du cas d'utilisation pour l'examen rétinien de l'enfant

Obtenir la signature rétinienne à partir de la base de données principale

Scannez la rétine de l'utilisateur et obtenez sa signature.

Comparer la signature originale à la signature scannée

Figure 13-189. Séquences comportementales pour les cas d'utilisation parents-enfants

L'enfant hérite de la séquence comportementale du parent et peut y insérer des comportements supplémentaires (Figure 13-189). Le parent et l'enfant sont potentiellement instanciables (s'ils ne sont pas abstraits) et représentent différentes spécialisations d'un même modèle. Les parents sont indépendants, contrairement à une relation d'extension, dans laquelle plusieurs extensions Tous modifient implicitement le même cas d'utilisation de base. Un comportement peut être ajouté à l'enfant. cas d'utilisation en ajoutant des étapes à la séquence de comportement héritée du parent, comme ainsi qu'en déclarant des liens de parenté **étendus** et **inclusifs** avec l'enfant. Si le parent est de manière abstraite, sa séquence de comportement peut comporter des sections explicitement incomplètes. Le parent doit fournir ces informations, qui doivent être fournies par l'enfant. L'enfant peut modifier les étapes héritées du parent, mais, comme pour la redéfinition des méthodes, cette capacité doit être soumise à autorisation. à utiliser avec précaution car l'intention du parent doit être préservée.

La relation de généralisation relie un cas d'utilisation enfant à un cas d'utilisation parent.

Le cas d'utilisation enfant peut accéder aux attributs définis par le cas d'utilisation parent et les modifier.

La substituabilité des cas d'utilisation signifie que la séquence de comportement d'un enfant utilise Le cas doit inclure la séquence comportementale de son parent. Les étapes de la séquence du parent ne doivent pas nécessairement être contiguës ; l'enfant peut y insérer des étapes supplémentaires. parmi les étapes de la séquence comportementale héritée du parent.

L'utilisation de l'**héritage multiple** avec des cas d'utilisation nécessite une spécification explicite de la manière dont les séquences comportementales des parents s'entremêlent pour former la séquence pour l'enfant.

La généralisation des cas d'utilisation peut recourir à l'**héritage privé** pour partager l'implémentation d'un cas d'utilisation de base sans substituabilité totale, mais cette capacité devrait être utilisé avec parcimonie.

Notation

On utilise le symbole de généralisation normal : une ligne continue reliant l'enfant au parent, avec une pointe de flèche triangulaire creuse sur la ligne touchant le symbole du parent.

Exemple

La [figure 13-188](#) présente le cas d'utilisation abstrait « Vérifier l'identité » et sa spécialisation en deux parties. cas d'utilisation concrets, dont le comportement est illustré dans [la figure 13-189](#).

instance de cas d'utilisation

L'exécution d'une séquence d'actions spécifiée dans un cas d'utilisation. Une instance d'un cas d'utilisation. cas.

Voir [le cas d'utilisation](#).

modèle de cas d'utilisation

Un modèle qui décrit les exigences fonctionnelles d'un système ou d'un autre classificateur en termes de cas d'utilisation.

Voir [acteur](#), [cas d'utilisation](#).

Sémantique

Le modèle de cas d'utilisation représente la fonctionnalité d'un système ou d'un autre classificateur telle qu'elle est manifestée aux utilisateurs externes interagissant avec le système. Un modèle de cas d'utilisation est présenté sur un écran [d'utilisation](#). [Schéma du boîtier](#).

vue de cas d'utilisation

Cet aspect du système concernait la spécification du comportement en termes d' [utilisation](#). [cas](#). Un [modèle de cas d'utilisation](#) est un [modèle](#) centré sur cette vue. La vue des cas d'utilisation fait partie de l'ensemble des concepts de modélisation regroupés de manière informelle sous l'appellation [de vue dynamique](#).

utilitaire

Un [stéréotype](#) de classe qui regroupe les variables et procédures globales sous la forme d'un Déclaration de classe. Les attributs et les opérations de l'utilitaire deviennent respectivement des variables globales et des procédures globales. Un utilitaire n'est pas un modèle fondamental. Il s'agit d'une construction, mais d'une commodité de programmation. Elle ne possède pas d'instances.

Sémantique

Les attributs de l'utilitaire sont des variables globales, et les opérations de l'utilitaire sont des opérations globales. Les utilitaires sont inutiles pour la programmation orientée objet. car les attributs et opérations globaux peuvent être mieux modélisés en tant que membres de portée classe.

Ce dispositif est prévu pour assurer la compatibilité avec les langages non orientés objet.
comme C.

Notation

Un utilitaire est représenté par un symbole de classe avec le mot-clé stéréotypé « utilitaire » au-dessus.
Nom de classe (chaîne de caractères). Les attributs et les opérations représentent des membres globaux. Non
Les membres de portée classe peuvent être déclarés dans le symbole.

valeur

Voir la valeur des données.

sommet

Une source ou une cible pour une transition dans une machine à états. Un sommet peut être soit une
État ou pseudo-État.

voir

Une projection d'un modèle, vu d'une perspective ou d'un point de vue particulier et
omet les entités qui ne sont pas pertinentes dans cette perspective. Le mot n'est pas utilisé ici pour
désigne un élément de présentation. Au contraire, il inclut des projections à la fois dans le modèle
sémantique et dans la notation visuelle.

visibilité

Une énumération dont la valeur (publique, protégée ou privée) indique si la
L'élément de modèle auquel il fait référence peut être vu en dehors de son espace de noms englobant.
Voir aussi la section « Accès » pour une discussion sur les règles de visibilité appliquées aux références inter-paquets.
ences.

Sémantique

La visibilité indique la capacité d'un élément de modélisation à référencer un élément qui est
dans un espace de noms différent de celui de l'élément référençant. La visibilité fait partie de la relation entre
un élément et le conteneur qui le contient. Le conteneur peut être
un package, une classe ou un autre espace de noms. Il existe trois niveaux de visibilité prédéfinis.

publique	Tout élément pouvant voir le conteneur peut également voir le élément indiqué.
protégé	Seul un élément contenu dans le conteneur ou un descendant de Le conteneur peut voir l'élément indiqué. Les autres éléments ne peuvent ni le référencer ni l'utiliser d'aucune autre manière.

privé Seul un élément contenu dans le conteneur peut voir l'élément. Les autres éléments, y compris les éléments descendants, ne peuvent pas voir cet élément. du conteneur, ne peut ni y faire référence ni l'utiliser d'aucune autre manière.

Des types de visibilité supplémentaires pourraient être définis pour certains langages de programmation. comme la visibilité de l'implémentation C++ (en fait, toutes les formes de visibilité non publique) (sont dépendantes du langage). L'utilisation d'options supplémentaires doit faire l'objet d'une convention entre l'utilisateur et les outils de modélisation et les générateurs de code.

Notation

La visibilité peut être indiquée par un mot-clé de propriété ou par un signe de ponctuation placé dans devant le nom d'un élément de modèle.

public	+
protégé	#
privé	-

L'indicateur de visibilité peut être désactivé. Son absence indique que la visibilité n'est pas affichée, et non qu'elle est indéfinie ou publique. Un outil Il convient d'attribuer une visibilité aux nouveaux éléments même si cette visibilité n'est pas affichée. Un marqueur de visibilité est un raccourci pour une chaîne de spécification complète de propriété de visibilité. La visibilité peut également être spécifiée par des mots-clés (public, protégé, privé). Ce formulaire est souvent utilisé comme élément de liste en ligne qui s'applique à un bloc entier d'attributs ou autres éléments de la liste.

Tout choix de visibilité spécifique à une langue ou défini par l'utilisateur doit être spécifié par un chaîne de propriétés ou selon une convention spécifique à l'outil.

Classes. Dans une **classe**, le marqueur de visibilité est placé sur les éléments **de liste** , tels que **les attributs**. et **les opérations**. Cela indique si une autre classe peut accéder aux éléments.

Associations. Dans une **association**, le marqueur de visibilité est placé sur le **nom du rôle** de la classe cible (l'extrémité accessible via le paramètre de visibilité). indique si la classe située à l'extrémité opposée peut parcourir l'association vers la fin avec le marqueur de visibilité.

Packages. Dans un **package**, le marqueur de visibilité est placé sur les éléments contenus directement dans le package, tels que les classes, les associations et les packages imbriqués. indique si un autre paquet qui accède au premier paquet ou l'importe peut voir les éléments.

bien formé

Désigne un modèle correctement construit, qui satisfait à toutes les règles et contraintes prédéfinies et spécifiées dans le modèle. Un tel modèle a une signification Sémantique. Un modèle mal formé est dit **mal formé**.



Éléments standard

Les éléments standard sont des mots-clés prédéfinis pour **les contraintes**, **les stéréotypes** et **les étiquettes**. Ces concepts représentent des notions d'utilité générale, insuffisamment significatives ou trop similaires aux concepts fondamentaux pour être intégrées à UML. Leur relation avec les concepts fondamentaux d'UML est comparable à celle d'une bibliothèque de sous-programmes intégrée à un langage de programmation : ils ne font pas partie intégrante du langage, mais constituent un élément essentiel de l'environnement sur lequel l'utilisateur peut compter. La liste comprend également des mots clés de notation, c'est-à-dire des mots clés apparaissant sur le symbole d'un autre élément du modèle, mais désignant des éléments intégrés et non des stéréotypes. Le symbole de notation correspondant à chaque mot clé est indiqué.

Les renvois renvoient à des articles du **chapitre 13, Encyclopédie des termes**.

accéder

(stéréotype de la dépendance à la permission)

Une dépendance stéréotypée entre deux paquets, indiquant que le contenu public du paquet cible est accessible à l'espace de noms du paquet source.

Voir **l'accès**.

association

(stéréotype de l'association)

Une contrainte appliquée à une extrémité d'association (y compris une extrémité de lien ou une extrémité d'un rôle d'association), spécifiant que l'instance correspondante est visible via une association réelle, plutôt que via un lien transitoire, tel qu'un paramètre ou une variable locale.

Voir **association**, **fin d'association**, **rôle de l'association**.

devenir

(stéréotype de la relation de flux)

Une dépendance stéréotypée dont la source et la cible représentent la même instance à différents moments, mais chacune avec des valeurs potentiellement différentes, est un état instance et rôles. Une dépendance de A à B signifie que l'instance A devient B avec éventuellement de nouvelles valeurs, un nouvel état et de nouveaux rôles à un instant différent. dans le temps et l'espace. La notation « devenir » est une flèche en pointillés allant de la source à la cible. avec le mot-clé «devenir» .

Voir **devenir**.

lier

(mot-clé sur le symbole de dépendance)

Un mot-clé de dépendance qui désigne une relation **de liaison** . Il est suivi d'un Liste **d'arguments** séparés par des virgules entre parenthèses.

Voir **liaison, élément lié, modèle**.

appel

(stéréotype de la dépendance à l'usage)

Une dépendance stéréotypée dont la source est une opération et la cible de qui est une opération. Une dépendance d'appel spécifie que la source invoque l'opération cible. Une dépendance d'appel peut connecter une opération source à n'importe quelle cible. opération qui entre dans le champ d'application, y compris, mais sans s'y limiter, les opérations du classificateur englobant et les opérations d'autres classificateurs visibles.

Voir **l'appel, l'utilisation**.

complet

(contrainte sur la généralisation)

Une contrainte appliquée à un ensemble de généralisations, spécifiant que tous les enfants ont ont été spécifiés (bien que certains puissent être omis) et il n'est pas prévu que d'autres enfants soient déclarés ultérieurement.

Voir **la généralisation**.

copie

(stéréotype de la relation de flux)

Une relation de flux stéréotypée dont la source et la cible sont des instances différentes, mais chacune ayant les mêmes valeurs, le même état et les mêmes rôles (mais un distinct

Une dépendance de copie de A vers B signifie que B est une copie exacte de A. Les modifications futures apportées à A ne sont pas nécessairement répercutées dans B. La notation de copie est une flèche en pointillés allant de la source à la cible, accompagnée du mot-clé « copy » .

Voir [l'accès, la copie](#).

créer

(stéréotype de caractéristique comportementale)

Une caractéristique comportementale stéréotypée indiquant que la caractéristique désignée crée une instance du classificateur auquel elle est rattachée.

(stéréotype de l'événement)

Un événement stéréotypé indiquant la création de l'instance contenant la machine à états à laquelle il s'applique. L'événement « Créer » ne peut être appliqué qu'à une transition initiale au niveau le plus élevé de cette machine à états. En fait, il s'agit du seul type de déclencheur pouvant être appliqué à une transition initiale.

(stéréotype de dépendance à l'usage)

La fonction Create est une dépendance stéréotypée indiquant que le classificateur client crée des instances du classificateur fournisseur.

Voir [création, utilisation](#).

dériver

(stéréotype de la dépendance à l'abstraction)

Une dépendance stéréotypée dont la source et la cible sont des éléments, généralement, mais pas nécessairement, du même type. Une dépendance dérivée indique que la source peut être calculée à partir de la cible. La source peut être implémentée pour des raisons de conception, comme l'efficacité, même si elle est logiquement redondante.

Voir [dérivation, élément dérivé](#).

détruire

(stéréotype de caractéristique comportementale)

Une caractéristique comportementale stéréotypée indiquant que la caractéristique désignée détruit une instance du classificateur auquel elle est attachée.

(stéréotype de l'événement)

Un événement stéréotypé indiquant que l'instance contenant la machine à états à laquelle le type d'événement s'applique est détruite.

Voir [la destruction](#).

détruit

(contrainte sur ClassifierRole et AssociationRole)

Indique qu'une instance du rôle existe au début de l'exécution de l'interaction englobante, mais qu'elle est détruite avant la fin de l'exécution.

Voir [rôle d'association](#), [rôle de classificateur](#), [collaboration](#), [destruction](#).

disjoint

(contrainte sur la généralisation)

Une contrainte appliquée à un ensemble de généralisations, spécifiant qu'un objet ne peut être une instance de plus d'une généralisation enfant de cet ensemble. Cette situation ne se présente qu'en cas d'héritage multiple.

Voir [la généralisation](#).

document

(stéréotype du composant)

Un composant stéréotypé représentant un document.

Voir [le composant](#).

documentation

(balise sur l'élément)

Un commentaire, une description ou une explication de l'élément auquel il est rattaché.

Voir [commentaire](#), [chaîne de caractères](#).

énumération

(mot-clé sur le symbole du classificateur)

Un mot-clé pour un type de données d'énumération, dont les détails spécifient un domaine constitué d'un ensemble d'identifiants qui sont les valeurs possibles d'une instance du type de données.

Voir [l'énumération](#).

exécutable

(stéréotype du composant)

Un composant stéréotypé désignant un programme pouvant être exécuté sur un nœud.

Voir [le composant](#).

étendre

(mot-clé sur le symbole de dépendance)

Un mot-clé sur un symbole de dépendance indiquant une relation étendue entre les utilisations cas.

Voir **étendre**.

façade

(stéréotype du paquet)

Un paquetage stéréotypé ne contenant que des références à des éléments de modèle appartenant à un autre paquetage. Il sert à fournir une vue publique d'une partie du contenu d'un paquetage.

Une façade ne contient aucun élément de modèle propre.

Voir **l'emballage**.

déposer

(stéréotype du composant)

Un fichier est un composant stéréotypé représentant un document contenant du code source ou des données.

Voir **le composant**.

cadre

(stéréotype du paquet)

Un ensemble stéréotypé composé principalement de motifs.

Voir **l'emballage**.

ami

(stéréotype de la dépendance à la permission)

Une dépendance stéréotypée dont la source est un élément du modèle (opération, classe ou package) et la cible un autre élément du modèle de package (classe ou package). Une relation d'amitié permet à la source d'accéder à la cible, indépendamment de la visibilité déclarée. Elle étend la visibilité de la source afin que la cible puisse voir son contenu.

Voir **accès, ami, visibilité**.

mondial

(stéréotype de l'association)

Une contrainte appliquée à une extrémité d'association (y compris les extrémités de liens et les extrémités de rôles d'association), spécifiant que l'objet attaché est visible car il se trouve dans un contexte global portée relative à l'objet situé à l'autre extrémité du lien.

Voir [association](#), [fin d'association](#), [collaboration](#).

mise en œuvre

(stéréotype de la généralisation)

Une généralisation stéréotypée indiquant que le client hérite de l'implémentation du fournisseur (ses attributs, ses opérations et ses méthodes) mais ne les rend pas publics. Les interfaces du fournisseur ne garantissent pas leur prise en charge, ce qui contrevient au principe de substituabilité. Il s'agit d' [un héritage privé](#).

Voir [généralisation](#), [héritage privé](#).

classe d'implémentation

(stéréotype de classe)

Une classe stéréotypée qui n'est pas un type et qui représente l'implémentation d'une classe dans un langage de programmation. Un objet peut être une instance d'au plus, une seule classe d'implémentation. En revanche, un objet peut être une instance de plusieurs classes ordinaires simultanément et peut acquérir ou perdre des classes au fil du temps. Une instance d'un La classe d'implémentation peut également être une instance de zéro ou plusieurs types.

Voir [la classe d'implémentation](#), [le type](#).

implicite

(stéréotype d'association)

Un stéréotype d'association, précisant que l'association n'est pas manifeste (mise en œuvre), mais seulement conceptuelle.

Voir [association](#).

importer

(stéréotype de la dépendance à la permission)

Une dépendance stéréotypée entre deux paquets, indiquant que le contenu public du paquet cible est ajouté à l'espace de noms du paquet source.

Voir [accès](#), [importation](#).

inclure

(mot-clé sur le symbole de dépendance)

Un mot-clé sur les symboles de dépendance indiquant une relation d'inclusion entre les utilisations cas.

Voir **inclure**.

incomplet

(contrainte sur la généralisation)

L'incomplétude est une contrainte appliquée à un ensemble de généralisations, précisant que tous les enfants n'ont pas été spécifiés et que d'autres enfants devraient être ajoutés.

Voir **la généralisation**.

instanceOf

(mot-clé sur le symbole de dépendance)

Une méta-relation dont le client est une instance et le fournisseur un classificateur. Une dépendance instanceOf de A vers B signifie que A est une instance de B.

La notation pour instanceOf est une flèche en pointillés avec le mot-clé «instanceOf».

Voir **descripteur, instance, instance de**.

instancier

(stéréotype de la dépendance à l'usage)

Une dépendance stéréotypée entre les classificateurs indiquant que les opérations sur le client créent des instances du fournisseur.

Voir **l'instanciation et l'utilisation**.

invariant

(stéréotype de la contrainte)

Une contrainte stéréotypée qui doit être associée à un ensemble de classificateurs ou de relations. Elle indique que les conditions de la contrainte doivent être vérifiées pour les classificateurs ou les relations et leurs instances.

Voir **invariant**.

feuille

(mot-clé sur GeneralizableElement et BehavioralFeature)

Indique un élément qui ne peut pas avoir de descendants ou qui ne peut pas être remplacé, c'est-à-dire un élément non polymorphe.

Voir [feuille](#), [polymorphe](#).

bibliothèque

(stéréotype du composant)

Un composant stéréotypé représentant une bibliothèque statique ou dynamique.

Voir [le composant](#).

locale

(stéréotype de l'association)

Un stéréotype d'extrémité d'association, d'extrémité de lien ou d'extrémité de rôle d'association, spécifiant que l'objet attaché se trouve dans une portée locale de l'objet à l'autre extrémité.

Voir [association](#), [fin d'association](#), [collaboration](#), [lien transitoire](#).

emplacement

(étiquette sur le symbole du classificateur)

Le composant qui prend en charge le classificateur.

(mot-clé sur le symbole d'instance du composant)

L'instance de nœud sur laquelle réside l'instance du composant.

Voir [composant](#), [emplacement](#), [nœud](#).

métaclasse

(stéréotype du classificateur)

Un classificateur stéréotypé indiquant que la classe est une métaclasse d'une autre classe.

Voir [la métaclasse](#).

nouveau

(contrainte sur ClassifierRole et AssociationRole)

Indique qu'une instance du rôle est créée lors de l'exécution de l'interaction englobante et existe toujours à la fin de l'exécution.

Voir [rôle d'association](#), [rôle de classificateur](#), [collaboration](#), [création](#).

chevauchement

(contrainte sur la généralisation)

Une contrainte appliquée à un ensemble de généralisations, spécifiant qu'un objet peut être une instance de plusieurs généralisations enfants de cet ensemble. Cette situation ne peut se produire qu'en cas d'héritage multiple ou de classification multiple.

Voir [la généralisation](#).

paramètre

(stéréotype de l'association)

Un stéréotype de fin d'association (y compris la fin d'un lien et la fin d'un rôle d'association), spécifiant qu'un objet attaché est un argument d'un appel à une opération sur l'objet à l'autre extrémité.

Voir [rôle d'association](#), [rôle de classificateur](#), [collaboration](#), [paramètre](#), [lien transitoire](#).

persistance

(balise sur Classificateur, Association et Attribut)

Indique si une valeur d'instance doit survivre au processus qui l'a créée. Les valeurs peuvent être persistantes ou transitoires. Appliqué aux attributs, il permet de mieux cibler les valeurs d'attributs à conserver au sein d'un classificateur.

Voir [l'objet persistant](#).

postcondition

(stéréotype de la contrainte)

Une contrainte stéréotypée qui doit être associée à une opération. Elle désigne les conditions qui doivent être respectées après l'appel de l'opération.

Voir [la postcondition](#).

type d'alimentation

(stéréotype du classificateur)

Un classificateur stéréotypé indiquant que le classificateur est une métaclasse dont les instances sont des sous-classes d'une autre classe.

(mot-clé sur le symbole de dépendance)

Une relation dont le client est un ensemble de généralisations et dont le fournisseur est un powertype. Le fournisseur est le powertype du client.

Voir [type d'alimentation](#).

condition préalable

(stéréotype de la contrainte)

Une contrainte stéréotypée qui doit être associée à une opération. Elle désigne les conditions qui doivent être remplies au moment de l'appel de l'opération.

Voir [la condition préalable](#).

processus

(stéréotype du classificateur)

Un classificateur stéréotypé qui est une classe active représentant un processus lourd.

Voir [la classe](#), [le processus](#) et [le thread actifs](#).

affiner

(stéréotype sur la dépendance à l'abstraction)

Un stéréotype de dépendance qui dénote une relation [de raffinement](#).

Voir [le raffinement](#).

exigence

(stéréotype de Commentaire)

Un commentaire stéréotypé qui énonce une responsabilité ou une obligation.

Voir [les exigences](#) et [les responsabilités](#).

responsabilité

(stéréotype dans les commentaires)

Un contrat ou une obligation du classificateur. Il est exprimé sous forme de chaîne de caractères.

Voir [les responsabilités](#).

soi

(stéréotype de l'association)

Un stéréotype de fin d'association (incluant la fin d'un lien et la fin d'un rôle d'association), spécifiant un pseudo-lien d'un objet vers lui-même afin d'appeler une opération sur ce même objet au sein d'une interaction. Cela n'implique pas une structure de données réelle.

Voir [rôle d'association](#), [rôle de classificateur](#), [collaboration](#), [paramètre](#), [lien transitoire](#).

sémantique

(étiquette sur le classificateur)

La spécification de la signification du classificateur.

(étiquette sur Opération)

La spécification de la signification de l'opération.

Voir [la sémantique](#).

envoyer

(stéréotype de la dépendance à l'usage)

Une dépendance stéréotypée dont le client est une opération ou un classificateur et le fournisseur un signal, précisant que le client envoie le signal à une cible non spécifiée.

Voir [envoyer, signaler](#).

stéréotype

(mot-clé sur le symbole du classificateur)

Mot-clé servant à définir un stéréotype. Ce nom peut être utilisé pour désigner d'autres éléments du modèle.

Voir [stéréotype](#).

bout

(stéréotype du paquet)

Un paquet stéréotypé représentant un paquet qui fournit les parties publiques d'un autre paquet, mais rien de plus.

Notez que ce terme est également utilisé en UML pour décrire [les transitions simulées](#).

Voir [l'emballage](#).

système

(stéréotype du paquet)

Un ensemble stéréotypé de modèles d'un système, le décrivant selon différents points de vue, non nécessairement disjoints ; il s'agit de la structure de niveau supérieur dans la spécification du système. Il contient également des relations et des contraintes entre les éléments de différents modèles. Ces relations et contraintes n'ajoutent aucune information sémantique aux modèles. Elles décrivent plutôt les relations entre les modèles eux-mêmes, par exemple le suivi des exigences et l'historique de développement.

Un système peut être réalisé par un ensemble de systèmes subordonnés, chacun décrit par ses propres un ensemble de modèles regroupés dans un package système distinct. Un package système ne peut être que contenu dans un package système.

Voir [emballage](#), [modèle](#), [système](#).

tableau

(stéréotype du composant)

Un composant stéréotypé représentant une table de base de données.

Voir [le composant](#).

fil

(stéréotype du classificateur)

Un classificateur stéréotypé qui est une classe active, représentant un flux léger de contrôle.

Notez que le mot est utilisé dans un sens plus large dans ce livre pour désigner toute personne indépendante. dent, lieu d'exécution simultané.

Voir [la classe active](#), [le fil de discussion](#).

tracer

(mot-clé sur la dépendance d'abstraction)

Un mot-clé associé à un symbole de dépendance qui indique une relation [de traçabilité](#).

Voir [la trace](#).

transitoire

(contrainte sur ClassifierRole et AssociationRole)

Indique qu'une instance du rôle est créée lors de l'exécution de l'interaction englobante, mais qu'elle est détruite avant la fin de l'exécution.

Voir [rôle d'association](#), [rôle de classificateur](#), [collaboration](#), [création](#), [destruction](#), [lien transitoire](#).

taper

(stéréotype de classe)

Une classe stéréotypée utilisée pour spécifier un domaine d'instances (objets), ainsi que les opérations applicables à ces objets. Un type ne peut contenir aucun des méthodes, mais il peut avoir des attributs et des associations.

Voir [la classe d'implémentation](#), [le type](#).

utiliser

(mot-clé sur le symbole de dépendance)

Un mot-clé relatif à la dépendance qui désigne une relation **d'utilisation** .

Voir **l'utilisation**.

utilitaire

(stéréotype du classificateur)

Un classificateur stéréotypé sans instances. Il décrit une collection nommée d'attributs et d'opérations non membres, tous de portée classe.

Voir **l'utilitaire**.

xor

(contrainte d'association)

Une contrainte appliquée à un ensemble d'associations liées à une même classe, spécifiant que tout objet de cette classe ne sera lié que par une seule de ces associations. Il s'agit d'une contrainte OU exclusif (et non OU inclusif).

Voir **association**.

Partie 4 : Annexes



Annexe A

Métamodèle UML



Documents de définition UML

L'UML est définie par un ensemble de documents publiés par la Gestion des Objets Groupe [UML-98]. Ces documents sont inclus sur le CD d'accompagnement de ce document. Ce chapitre explique la structure du modèle sémantique UML décrit dans le livre. les documents.

UML est formellement défini à l'aide d'un métamodèle, c'est-à-dire un modèle des constructions UML. Le métamodèle lui-même est exprimé en UML. Ceci est un exemple de Interprète métacirculaire — c'est-à-dire un langage défini en fonction de lui-même. Les choses sont Pas totalement circulaire. Seul un petit sous-ensemble d'UML est utilisé pour définir le méta-modèle. En principe, ce point fixe de la définition pourrait être initialisé à partir de Une définition plus simple. En pratique, il est inutile d'aller aussi loin.

Chaque section du document sémantique contient un diagramme de classes illustrant une partie du métamodèle ; une description textuelle des classes du métamodèle définies dans ce diagramme. La section contient les attributs et les relations des éléments ; une liste des contraintes sur les éléments exprimées en langage naturel et en OCL ; et une description textuelle de la sémantique dynamique des constructions UML définies dans la section. La sémantique dynamique est : donc informel, mais une description entièrement formelle serait à la fois impraticable et Illisible par la plupart.

La notation est décrite dans un chapitre distinct qui fait référence au chapitre sur la sémantique. et associe les symboles aux classes du métamodèle.

Structure du métamodèle

Le métamodèle est divisé en trois paquets principaux (Figure A-1).

- Le package de base définit la structure statique de l'UML.
- Le module des éléments comportementaux définit la structure dynamique de l'UML.
- Le progiciel de gestion de modèles définit la structure organisationnelle d'UML modèles.

Pack de base

Le package de base contient quatre sous-packages.

Cœur

Le package principal décrit les principales constructions statiques d'UML. Celles-ci incluent **les classificateurs**, leur contenu et leurs relations. Leur contenu comprend **les attributs**, **les opérations**, **les méthodes** et **les paramètres**. Leurs relations incluent **la généralisation**, **l'association** et **la dépendance**. Plusieurs métaclasses abstraites sont également définies, telles que **l'élément généralisable**, **l'espace de noms** et **l'élément de modèle**. Le package définit également **les modèles** et divers types de sous-classes **de dépendance**, ainsi que **les composants**, **les nœuds** et **les commentaires**.

Types de données

Le package des types de données décrit les classes **de types de données** utilisées dans le métamodèle.

mécanismes d'extension

Le module des mécanismes d'extension décrit les mécanismes **de contrainte**, **de stéréotype** et **de valeur étiquetée**.

Module Éléments comportementaux Le

module comportemental comporte un sous-module pour chaque vue principale, ainsi qu'un module pour les constructions comportementales partagées par les trois vues principales.

Comportement courant

Le module de comportement commun décrit **les signaux**, **les opérations** et **les actions**. Il décrit également les classes **d'instances** correspondant à différents descripteurs.

Collaborations

Le module collaborations décrit **la collaboration**, **l'interaction**, **les messages**, **le rôle du classificateur** et **l'association**.

Cas d'utilisation

Le package de cas d'utilisation décrit **l'acteur** et **le cas d'utilisation**.

Machines à états

Le module « machines à états » décrit la structure **des machines à états**, notamment **les états** et divers types de **pseudo-états**, **d'événements**, **de signaux**, **de transitions** et **de conditions de garde**. Il décrit également des constructions supplémentaires pour les modèles d'activité, telles que **l'état d'action**, **l'état d'activité** et **l'état de flux d'objets**.

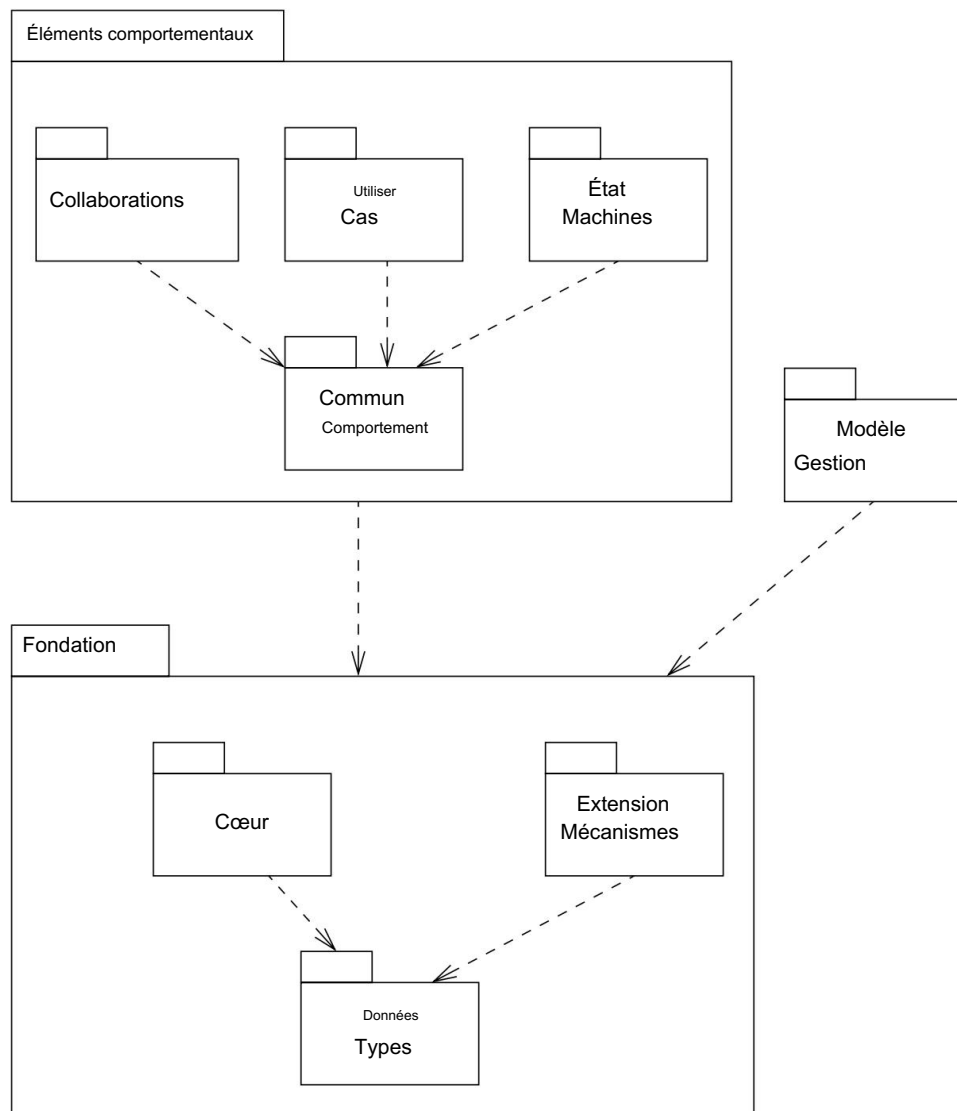


Figure A-1. Structure du métamodèle UML

Le package de gestion des modèles

décrit **les packages, les modèles et les sous-systèmes**. Il décrit également les propriétés de propriété et de visibilité des **espaces de noms et des packages**. Il ne contient aucun sous-package.

Annexe B

Résumé de la notation



Ce chapitre présente un bref aperçu visuel de la notation. Les principaux éléments de notation y sont inclus, mais toutes les variantes et options ne sont pas illustrées. Pour plus de détails, veuillez consulter l'article encyclopédique correspondant à chaque élément.

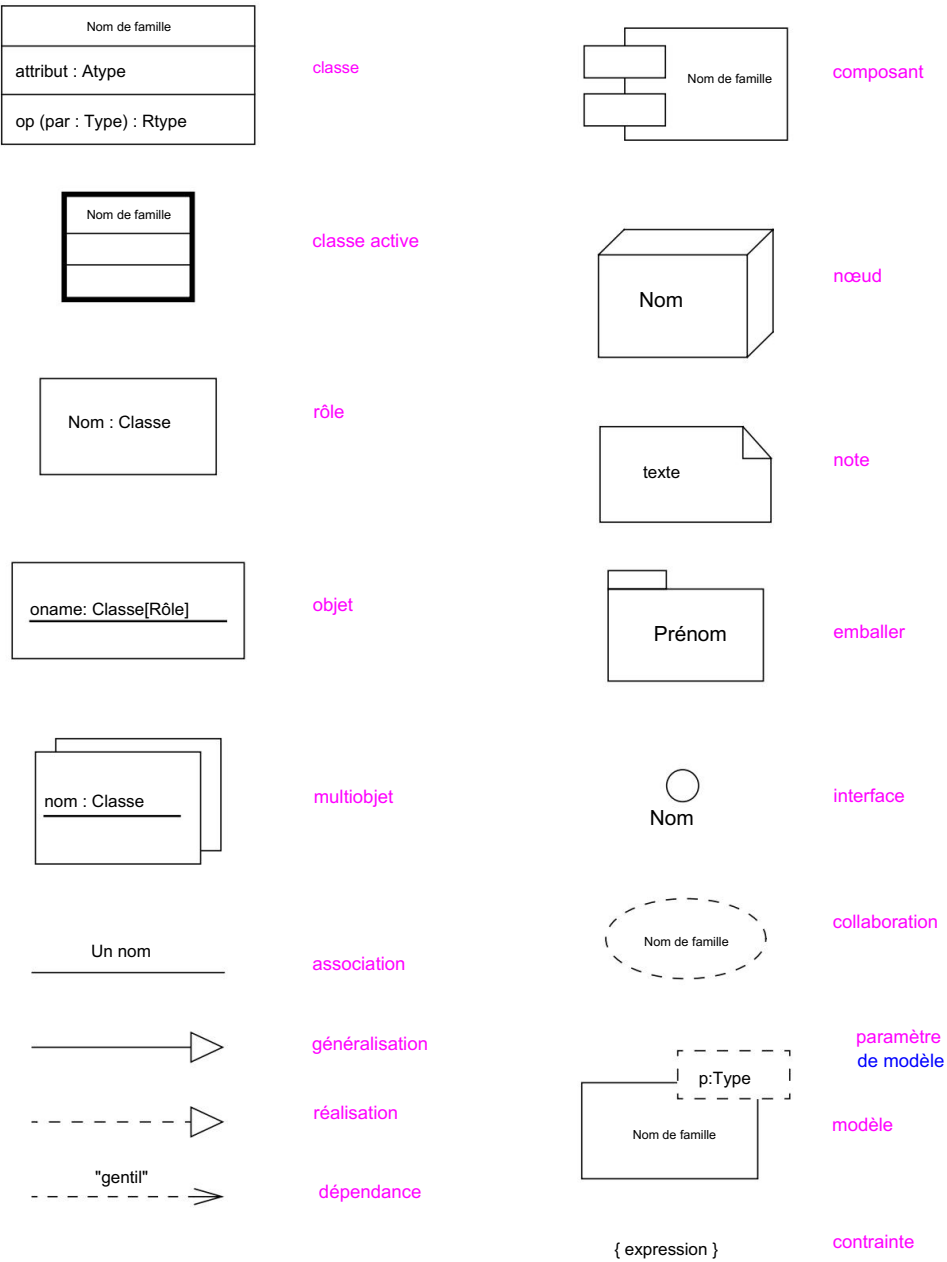


Figure B-1. Icônes sur les diagrammes de classe, de composant, de déploiement et de collaboration

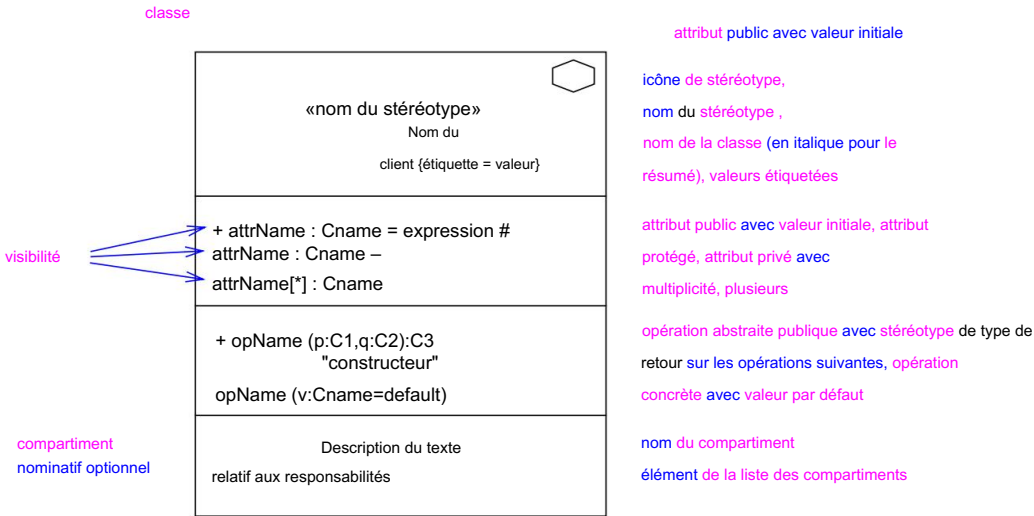


Figure B-2. Contenu du cours

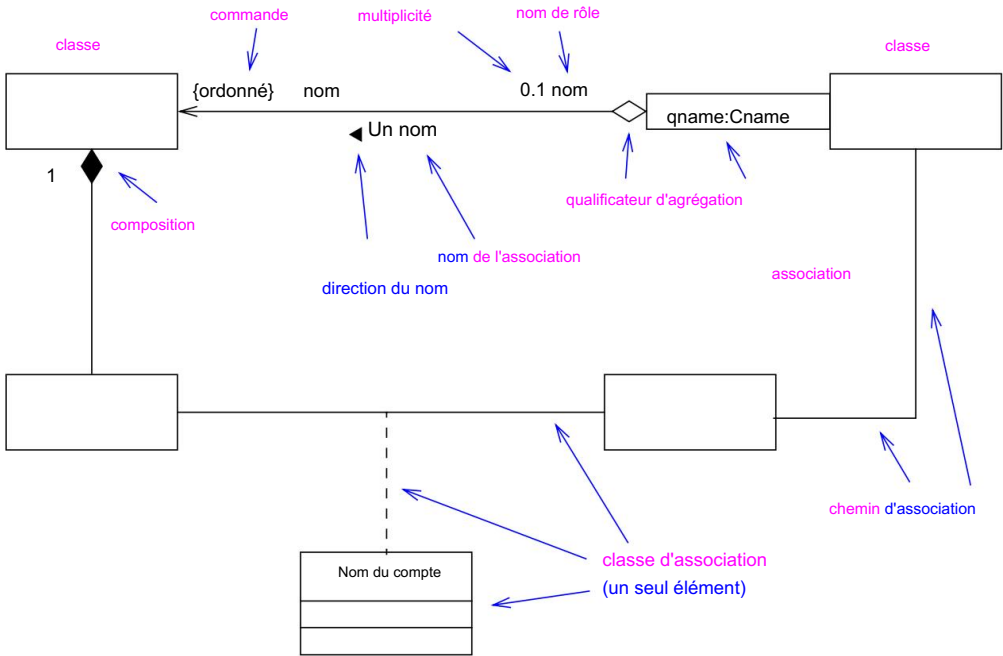


Figure B-3. Ornaments d'association dans un diagramme de classe

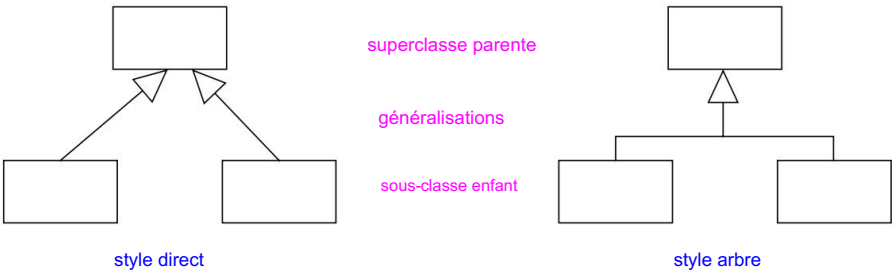


Figure B-4. Généralisation

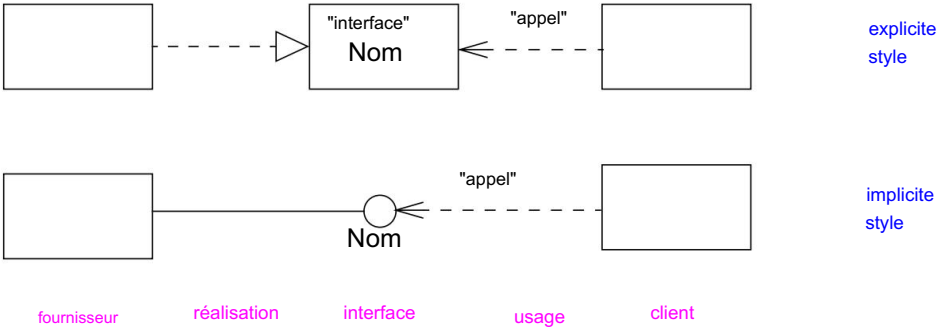


Figure B-5. Réalisation d'une interface

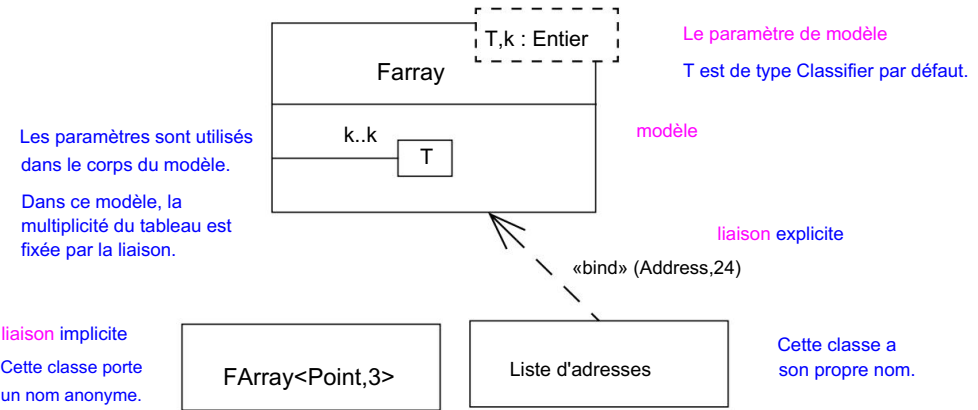


Figure B-6. Modèle

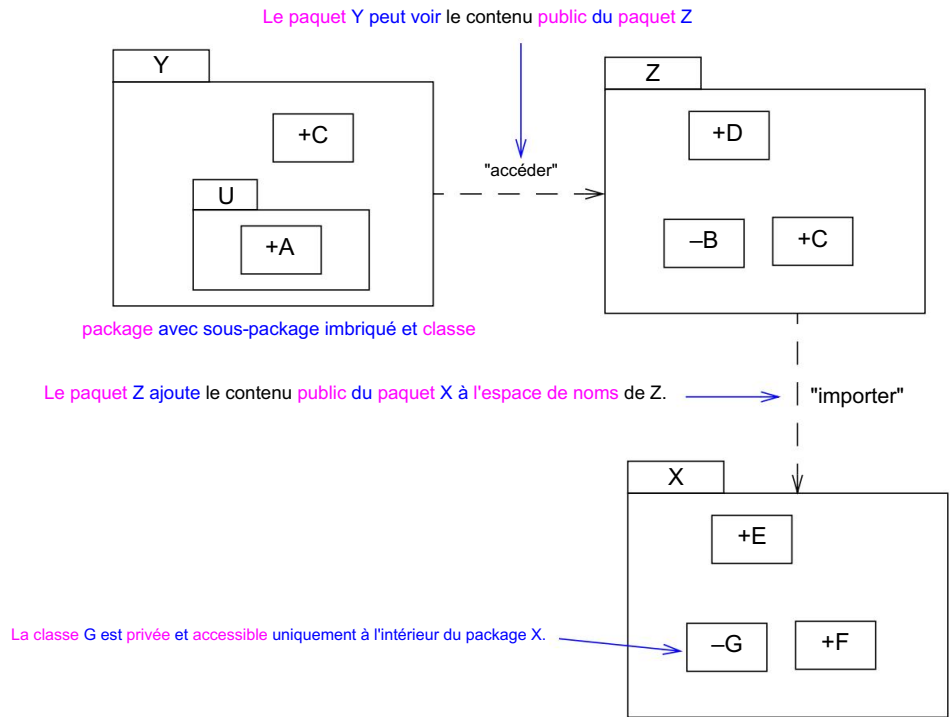


Figure B-7. Notation de l'emballage

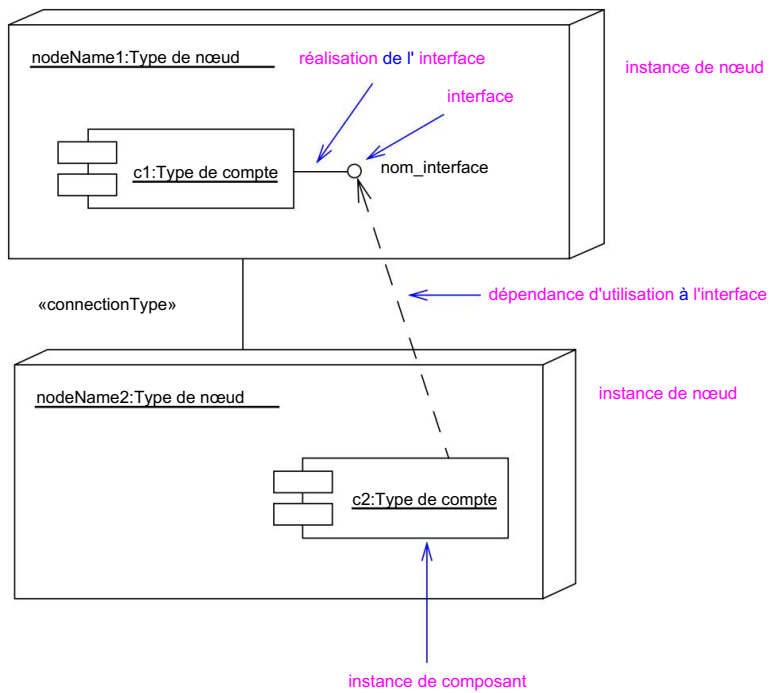


Figure B-8. Notation des composants et des nœuds

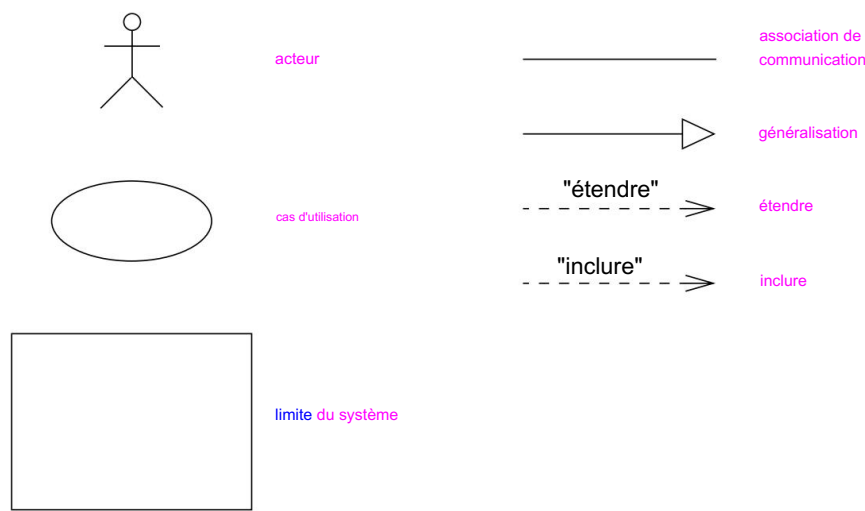


Figure B-9. Icônes sur les diagrammes de cas d'utilisation

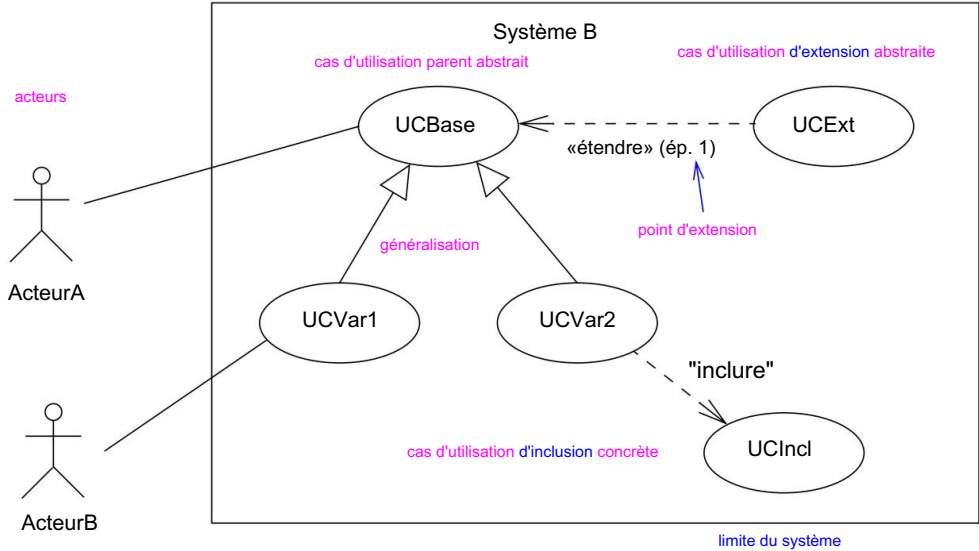


Figure B-10. Notation du diagramme de cas d'utilisation

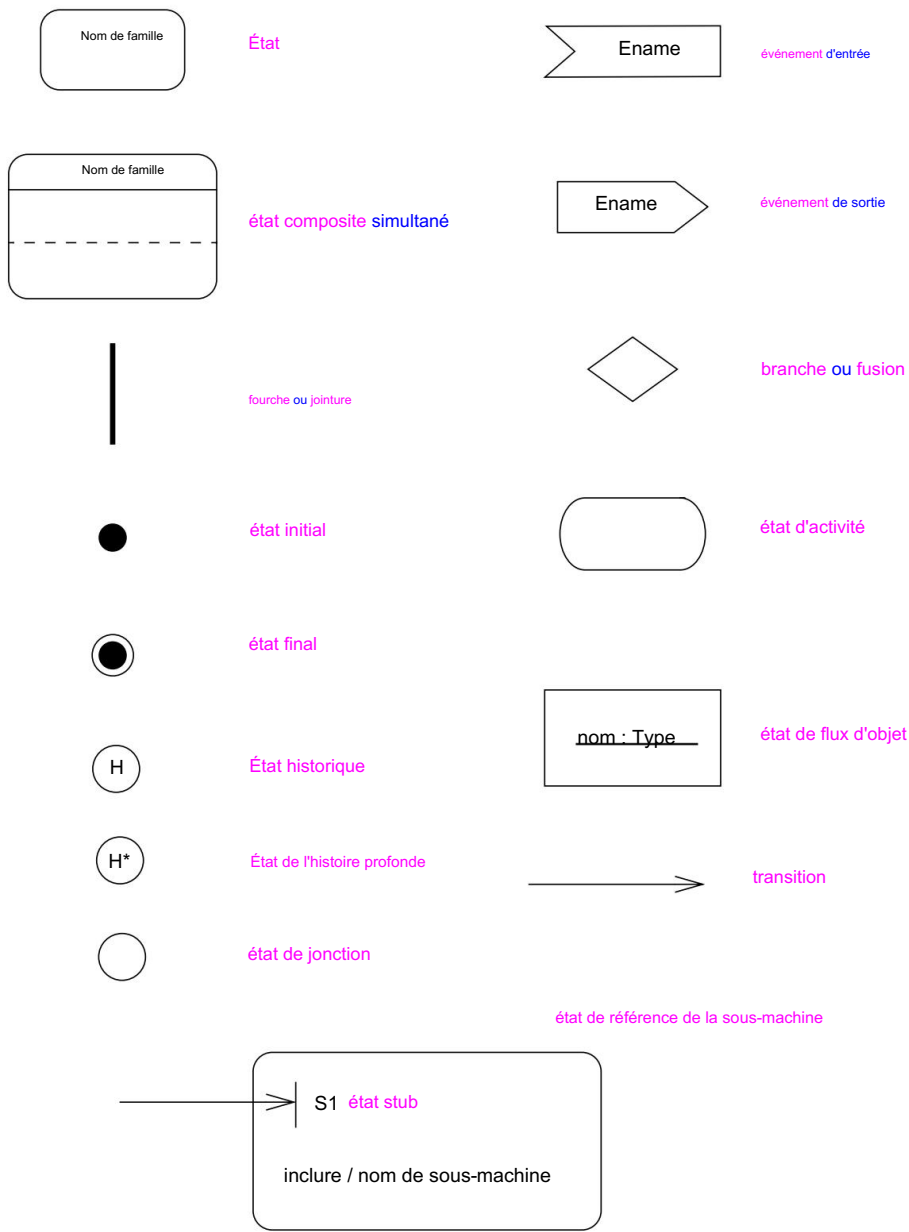


Figure B-11. Icônes sur les diagrammes d'états et d'activités

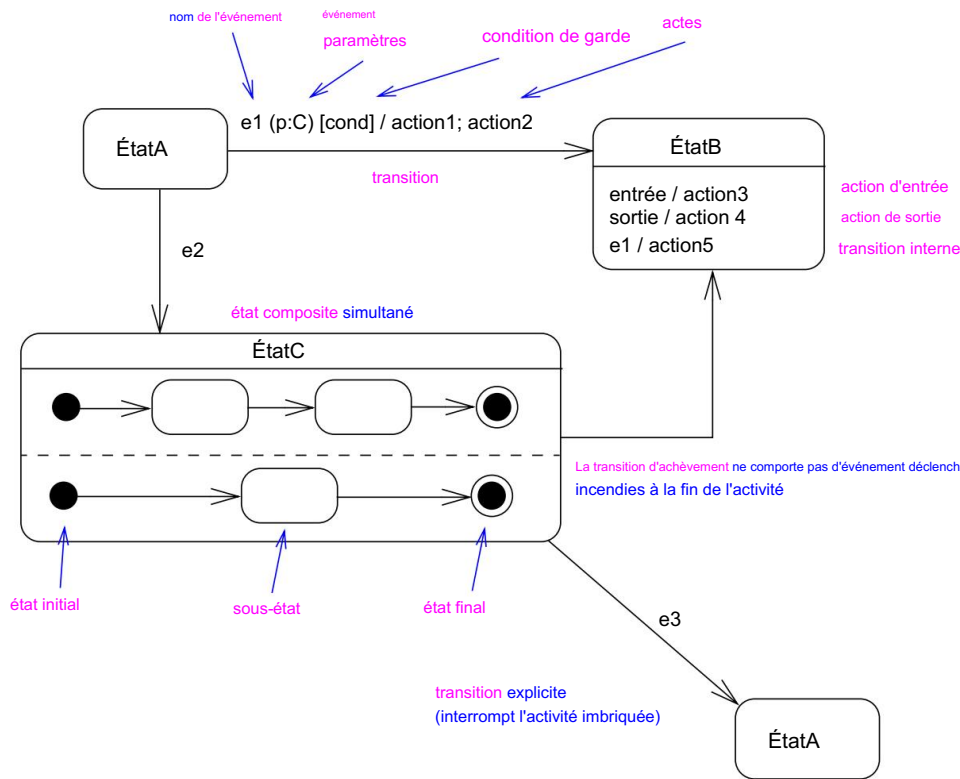


Figure B-12. Notation Statechart

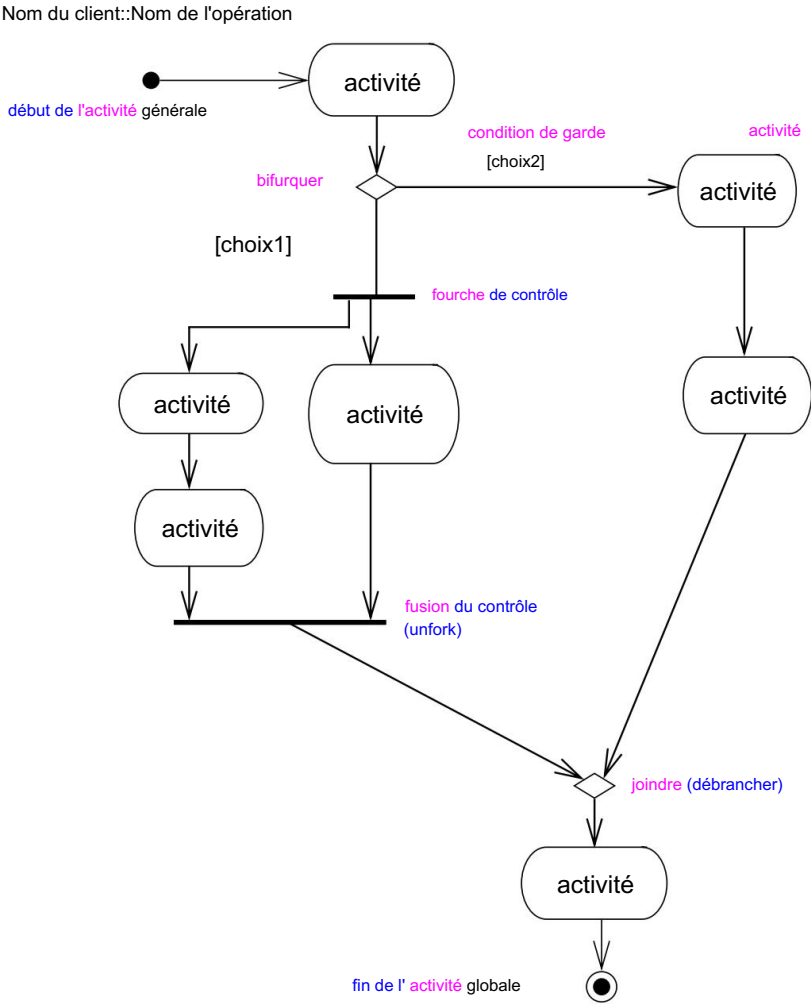


Figure B-13. Notation du diagramme d'activité

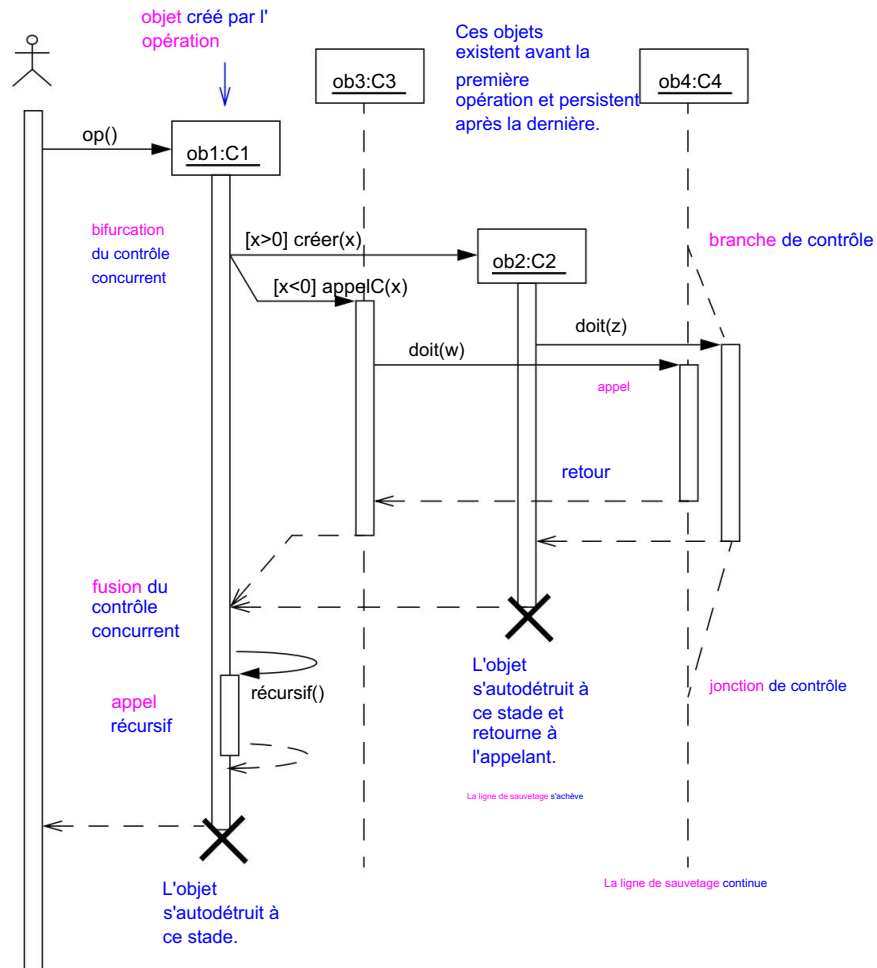


Figure B-14. Notation du diagramme de séquence

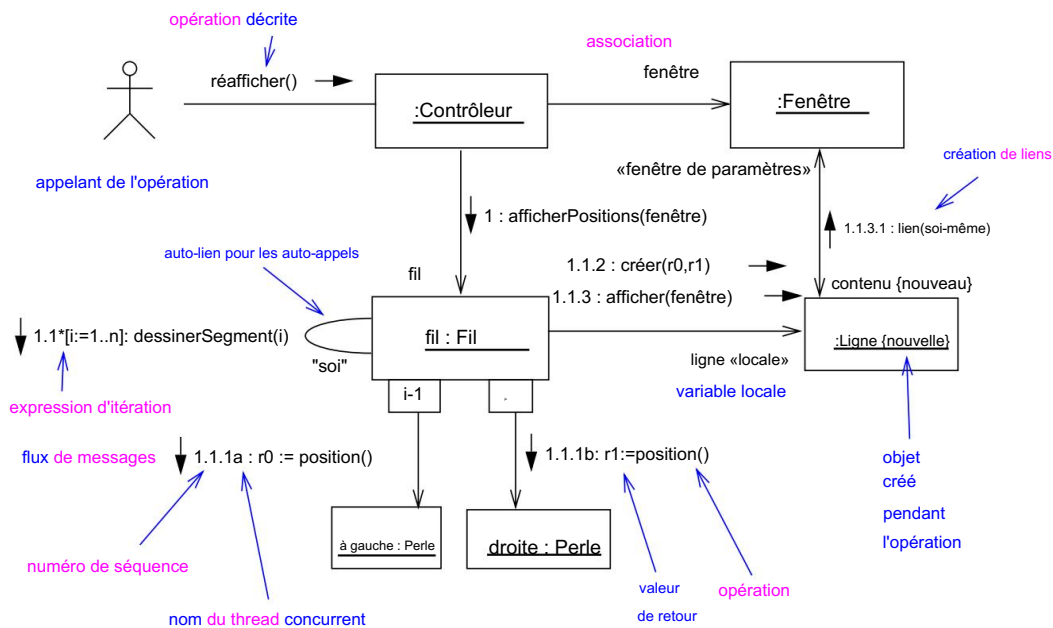


Figure B-15. Notation du diagramme de collaboration

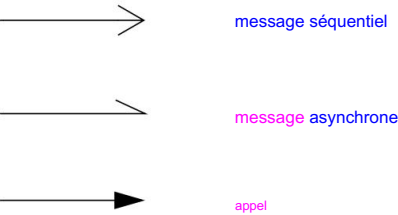


Figure B-16. Notation du message

Annexe C

Extensions de processus



Adaptation de l'UML

UML est conçu pour être utilisé à de nombreuses fins. Bien qu'UML soit un langage universel capable d'exprimer un grand nombre de propriétés de modélisation fondamentales,

Il existe des situations où un langage adapté à un domaine particulier est souhaitable.

Dans les langues naturelles, le jargon, l'argot et les vocabulaires spécialisés facilitent souvent la communication.

La communication au sein d'un métier ou d'une discipline spécialisée est facilitée, mais elle est entravée avec les personnes extérieures à ce domaine. L'UML peut être adaptée de manière similaire, à l'aide de divers mécanismes, tels que :

Conventions de nommage, règles de style, classes prédéfinies et correspondances implicites avec les logiciels. En

particulier, une extension UML peut être définie à l'aide **de stéréotypes prédéfinis.**

contraintes et **valeurs étiquetées.**

UML comporte de nombreuses constructions expressives, une extension spécialisée serait donc nécessaire.

À éviter sauf en cas d'absolue nécessité. Par définition, une extension a une utilité limitée.

au sein de la communauté, ce qui peut engendrer des malentendus et de la confusion chez les personnes extérieures.

Néanmoins, si une communauté cible est étroitement définie, le pouvoir expressif d'une

L'ajout d'extensions peut parfois compenser la perte d'uniformité. Avec le temps, certaines extensions peuvent

s'avérer suffisamment utiles pour être intégrées à la norme UML ; d'autres peuvent tomber en désuétude.

Ce chapitre décrit deux extensions fournies avec les documents UML

eux-mêmes. Leur utilisation n'est pas obligatoire ni même nécessairement recommandée, mais

Elles indiquent comment définir une extension pour un processus de développement. Plusieurs extensions similaires ont été définies dans la littérature.

Extensions du processus de développement logiciel

Ces extensions sont basées sur le processus de développement Objectory, un précurseur de

le processus unifié. Ils sont destinés à être utilisés lors d'un processus de développement logiciel qui comprend

quatre étapes : la capture des cas d'utilisation, l'analyse, la conception et l'implémentation.

tation.

Ces extensions comprennent des stéréotypes sur les unités d'emballage, les classes et les associations, ainsi

que certaines contraintes sur la connexion des éléments. Il n'y a pas de nouvelles balises.

Tableau C-1 : Stéréotypes organisationnels du processus de développement logiciel

Classe de base	Stéréotypes selon l'étape du processus			
	Cas d'utilisation	Analyse	Conception	Mise en œuvre
modèle	modèle d'analyse de cas d'utilisation		modèle de conception	modèle de mise en œuvre
emballer	système de cas d'utilisation cas d'utilisation emballer			système de mise en œuvre sous-système d'implémentation
sous-système		système d'analyse sous-système d'analyse forfait de service d'analyse	système de conception sous-système de conception forfait de services de conception	

Stéréotypes organisationnels

Le **tableau C-1** présente les stéréotypes organisationnels, c'est-à-dire les stéréotypes sur **le modèle**, **paquetage** et **sous-système**. Il existe divers stéréotypes applicables à chaque étape de développement. Dans le processus Objectory, chaque étape possède un modèle distinct, avec **une trace** relations entre les éléments dans différents modèles. De haut en bas, les termes Les termes système, sous-système et package de services décrivent des niveaux d'encapsulation.

Les stéréotypes organisationnels ne sont pas représentés par des icônes spécifiques. Ils sont représentés comme icônes de dossier, avec le nom stéréotypé en guillemets, comme indiqué dans **la figure C-1**.

Stéréotypes de classe

Trois stéréotypes sont définis concernant **la classe** : contrôle, frontière et entité. Ce sont illustré à **la figure C-2**.

Une classe de contrôle décrit les objets qui gèrent les interactions, telles qu'une transaction. Un gestionnaire, un contrôleur de périphérique ou un moniteur de système d'exploitation possède un comportement spécifique à un cas d'utilisation. Une classe de contrôle ne survit généralement pas au cas d'utilisation qu'elle prend en charge. Elle est représentée par un cercle surmonté d'une flèche.

Une classe frontière décrit les objets qui servent d'intermédiaires entre un système et l'extérieur. acteurs, tels qu'un formulaire de saisie de commande ou un capteur. Il est affiché sous la forme d'un cercle avec un Une barre en T y est fixée. Les objets limites existent souvent pendant toute la durée de vie d'un système.

Une classe d'entité décrit des objets passifs. Ils n'initient pas d'interactions. Les objets d'entité peuvent intervenir dans de nombreux cas d'utilisation et ont généralement une durée de vie supérieure à celle d'une seule entité. interactions. Cela se présente sous la forme d'un cercle avec une ligne en dessous.

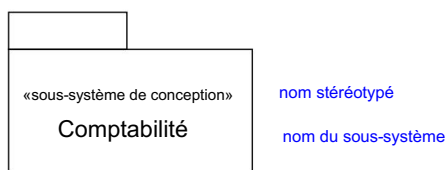


Figure C-1. Notation stéréotypée organisationnelle du processus de développement logiciel

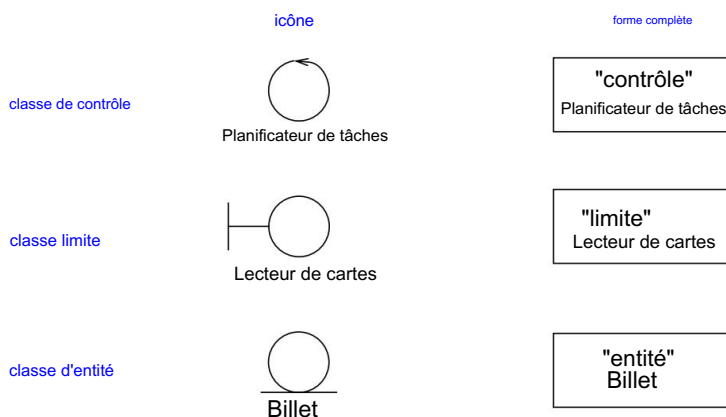


Figure C-2. Stéréotypes de classe pour le processus de développement logiciel

stéréotypes d'association

Il existe deux stéréotypes d' **association** : communiquer et s'abonner. La notation est un chemin d'association avec le nom du stéréotype en guillemets.

Une association de communication relie un **acteur** à un **cas d'utilisation** avec lequel il communique. Il s'agit de la seule association entre acteurs et cas d'utilisation ; le mot-clé peut donc être omis.

Une association d'abonnement relie une classe client (l'abonné) à une classe fournisseur (l'éditeur). L'abonné spécifie un ensemble d'événements que l'éditeur peut produire. L'abonné est averti lorsqu'un de ces événements se produit.

Extensions de modélisation d'entreprise

Ces extensions sont destinées à la modélisation d'organisations commerciales réelles et pour comprendre les situations réelles, plutôt que la mise en œuvre de logiciels. Ces stéréotypes ne sont pas nécessaires à la modélisation d'entreprise, mais ils en couvrent certains aspects. quelques situations courantes.

Ces extensions comprennent des stéréotypes sur les unités d'emballage, les classes et les associations, ainsi que certaines contraintes sur la connexion des éléments. Il n'y a pas de nouvelles balises.

Stéréotypes organisationnels

Le tableau C-2 présente les stéréotypes organisationnels pour la modélisation des processus métier.

Le modèle de cas d'utilisation est un modèle de la vue des cas d'utilisation. Le système de cas d'utilisation et l'utilisation Le contenu d'un dossier est organisé en deux niveaux.

Le modèle objet est un modèle de la structure interne du système d'information de l'entreprise. Le système d'objets est son sous-système de niveau supérieur qui contient les unités organisationnelles et Les unités de travail constituent les niveaux inférieurs. Une unité organisationnelle correspond à une unité organisationnelle de l'entreprise elle-même, et une unité de travail est une unité plus petite, mais significative. regroupement.

Tableau C-2 : Stéréotypes organisationnels pour la modélisation d'entreprise

Capture du cas d'utilisation	utilisation de la classe de base	Modèle objet
modèle	modèle de cas d'utilisation	modèle objet
emballer	système de cas d'utilisation package de cas d'utilisation	
sous-système		système d'objets unité organisationnelle unité de travail

Il n'existe pas d'icônes spécifiques pour les unités organisationnelles de la modélisation d'entreprise ; elles utilisent le symbole du dossier avec un stéréotype en guillemets.

Stéréotypes de classe

Outre les acteurs (définis dans la norme UML), les objets métier possèdent plusieurs stéréotypes de classe — travailleur, assistant social, agent interne, entité. Ils sont montrés dans la figure C-3.

Un travailleur social représente une personne qui agit au sein du système. Un intervenant social est un Un travailleur qui interagit directement avec des acteurs externes. Un travailleur interne est un travailleur qui interagit avec les travailleurs et les entités au sein du système.

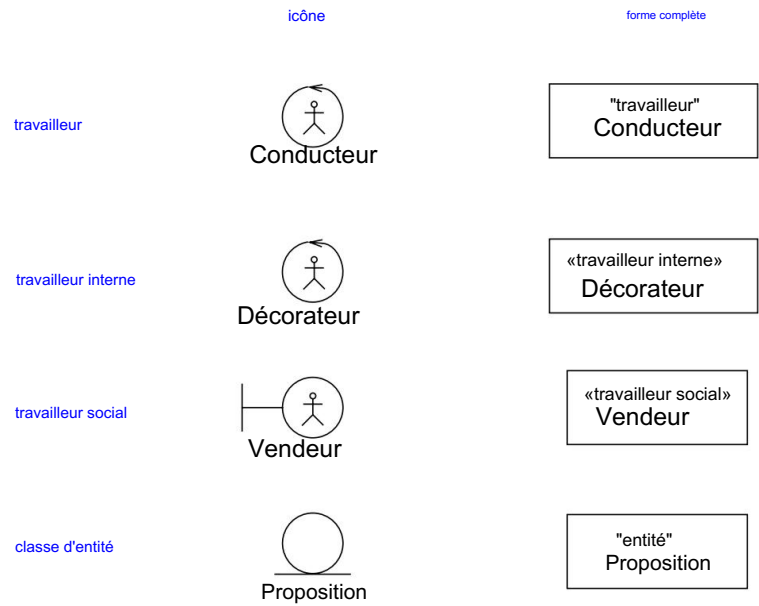


Figure C-3. Stéréotypes de classe pour la modélisation d'entreprise

Une classe d'entité décrit des objets passifs, qui n'initient pas d'interactions. Les objets d'entité peuvent intervenir dans de nombreux cas d'utilisation et survivent généralement à une seule interaction. Dans un contexte professionnel, les entités représentent généralement des produits de travail.

stéréotypes d'association

Il existe deux stéréotypes d'association : communiquer et s'abonner. Ils sont identiques à ceux du processus de développement logiciel. La notation est un chemin d'association avec le mot-clé stéréotype entre guillemets.

Une association de communication relie un **acteur** à un **cas d'utilisation** avec lequel il communique.

Il s'agit de la seule association entre acteurs et cas d'utilisation ; le mot-clé peut donc être omis.

Une association d'abonnement relie une classe cliente (l'abonné) à une classe fournisseur (l'éditeur).

L'abonné spécifie un ensemble d'événements que l'éditeur peut produire. L'abonné est notifié lorsqu'un de ces événements se produit.

Bibliographie



- [Blaha-98] Michael Blaha, William Premerlani. Modélisation et conception orientées objet pour les applications de bases de données. Prentice Hall, Upper Saddle River, NJ, 1998.
- [Booch-91] Grady Booch. Analyse et conception orientées objet avec applications, 1re éd. Benjamin/Cummings, Redwood City, Californie, 1991.
- [Booch-94] Grady Booch. Analyse et conception orientées objet avec applications, 2e éd. Benjamin/Cummings, Redwood City, Californie, 1994.
- [Booch-96a] Grady Booch. Object Solutions : Gestion de projet orienté objet. Addison-Wesley, Menlo Park, Californie, 1996.
- [Booch-96b] Grady Booch. Le meilleur de Booch : Stratégies de conception pour la technologie des objets. SIGS Books. New York, NY, 1996.
- [Booch-99] Grady Booch, James Rumbaugh, Ivar Jacobson. Guide de l'utilisateur du langage de modélisation unifié. Addison-Wesley, Reading, Mass., 1999.
- [Buschmann-96] Frank Buschmann, Régine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. Architecture logicielle orientée modèles : un système de modèles. Wiley, Chichester, Royaume-Uni, 1996.
- [Coad-91] Peter Coad, Edward Yourdon. Analyse orientée objet, 2e éd. Yourdon Press, Englewood Cliffs, NJ, 1991.
- [Coleman-94] Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes, Paul Jeremaes. Développement orienté objet : la méthode de fusion. Prentice Hall, Englewood Cliffs, NJ, 1994.
- [Cox-86] Brad J. Cox. Programmation orientée objet : une approche évolutionnaire. Addison-Wesley, Reading, Mass., 1986.
- [Embley-92] Brian W. Embley, Barry D. Kurtz, Scott N. Woodfield. Analyse des systèmes orientés objet : une approche basée sur les modèles. Yourdon Press, Englewood Cliffs, NJ, 1992.
- [Gamma-95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Mass., 1995.

- [Goldberg-83] Adele Goldberg, David Robson. Smalltalk-80 : Le langage et son implémentation. Addison-Wesley, Reading, Mass., 1983.
- [Harel-98] David Harel, Michal Politi. Modélisation des systèmes réactifs avec des diagrammes d'états : l'approche STATEMATE. McGraw-Hill, New York, NY, 1998.
- [Jacobson-92] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Övergaard. Génie logiciel orienté objet : une approche basée sur les cas d'utilisation. Addison-Wesley, Wokingham, Angleterre, 1992.
- [Jacobson-95] Ivar Jacobson, Maria Ericsson, Agneta Jacobson. L'avantage de l'objet : la réingénierie des processus métier grâce à la technologie objet. Addison-Wesley, Wokingham, Angleterre, 1995.
- [Jacobson-97] Ivar Jacobson, Martin Griss, Patrik Jonsson. Réutilisation du logiciel : architecture, processus et organisation pour la réussite des entreprises. Addison-Wesley, Harlow, Angleterre, 1997.
- [Jacobson-99] Ivar Jacobson, Grady Booch, James Rumbaugh. Le développement logiciel unifié Procédé. Addison-Wesley, Reading, Mass., 1999.
- [Martin-92] James Martin, James Odell. Analyse et conception orientées objet. Prentice Hall, Englewood Cliffs, NJ, 1992.
- [Meyer-88] Bertrand Meyer. Construction de logiciels orientés objet. Prentice Hall, New York, NY, 1988.
- [Rumbaugh-91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen. Modélisation et conception orientées objet. Prentice Hall, Englewood Cliffs, NJ, 1991.
- [Rumbaugh-96] James Rumbaugh. Perspectives OMT : Perspectives sur la modélisation tirées du Journal of Technologie orientée objet. SIGS Books, New York, NY, 1996.
- [Rumbaugh-99] James Rumbaugh, Ivar Jacobson, Grady Booch. Manuel de référence du langage de modélisation unifié. Addison-Wesley, Reading, Mass., 1999.
- [Selic-94] Bran Selic, Garth Gullekson, Paul T. Ward. Modélisation orientée objet en temps réel. Wiley, New York, NY, 1994.
- [Shlaer-88] Sally Shlaer, Stephen J. Mellor. Analyse des systèmes orientés objet : Modéliser le monde dans Données. Yourdon Press, Englewood Cliffs, NJ, 1988.
- [Shlaer-92] Sally Shlaer, Stephen J. Mellor. Cycles de vie des objets : modélisation du monde en états. Yourdon Presse, Englewood Cliffs, NJ, 1992.
- [UML-98] Spécification du langage de modélisation unifié. Groupe de gestion des objets, Framingham, Mass., 1998. Internet : www.omg.org.
- [Ward-85] Paul Ward, Stephen J. Mellor. Développement structuré pour les systèmes temps réel : Introduction et outils. Yourdon Press, Englewood Cliffs, NJ, 1985.
- [Warmer-99] Jos B. Warmer, Anneke G. Kleppe. Le langage de contraintes d'objet : une modélisation précise avec UML. Addison-Wesley, Reading, Mass., 1999.
- [Wirfs-Brock-90] Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener. Conception de logiciels orientés objet. Prentice Hall, Englewood Cliffs, NJ, 1990.
- [Yourdon-79] Edward Yourdon, Larry L. Constantine. Conception structurée : Fondements d'une discipline de conception de programmes et de systèmes informatiques. Yourdon Press, Englewood Cliffs, NJ, 1979.

Indice



Les entrées principales sont en gras ; les références aux figures sont en italique.

UN

temps absolu 476

abstrait 113, 286

classe 117

notation 193

opération 117

superclasses abstraites règle 115

abstraction 118

accès 98, 119, 379

action 73, 122

expression 126

implicite 71

langage 124–125

séquence 126

état 126

tableau de 74

sur la transition 480

activation 88, 88, 127 actif

129 classe

130 objet

131

configuration d'état actif 129, 133, 441 activité 133

diagramme 32,

81, 82, 83, 134 expression 134

graphique 135

état 142

vue 31, 81, 144

acteur 63, 144

paramètre réel, voir argument addOnly

(mot-clé) 185 agrégat 146

agrégation 146

composition 226

allocation de mémoire

257

responsabilité de

227 analyse 150 temps 150

ancêtre 150

architecture 150

argument 151

Arnold, Patrick 537

artefact 152
assertion 317, 391–392
affectation 123
association 47, 47, 49, 152 classe
 48, 157 à la
 classe dans l'état 191 fin
 160

 généralisation 163 mot-
 clé 478 n-aire 350
 efficacité de
 navigation 357 qualifié 398 rôle
 164 action
 asynchrone

165 contrôle 425

 événement 440
 message 336
atomique 165
attribut 166

 en tant que partie composite 228, 230

B

Informations générales 170 Message
d'erreur 336 Cas d'utilisation
de base 272, 297 deviennent
90, 171 Comportement
172

 séquence 491
 spécification 493
éléments

 comportementaux paquetage
 516 fonctionnalité
 172 vue 172

bidirectionnalité 50
association binaire 172 liaison
173 liaison

173 événement
de naissance 419

Blaha, Michael 5, 537–538
Bock, Conrad 7
Bodoff, Stéphanie 537
Booch, Grady 5–7, 537–538
Méthode Booch 5, 8 livres,
orienté objet 5
Expression

 booléenne 174 ,
élément lié 174 , branche
174, notation de diagramme de

 séquence 335 , notation d'état-transition
 240, 480, diffusion du signal 427

Brodsky, Steve 6
Buschmann, Frank 537

extensions de
 modélisation

 d'entreprise 534 stéréotypes,
tableau de 534 organisation d'entreprise comme couloir 462

C

C++ 5, 168–169, 216, 228, 236, 245,
 248–249, 271, 311, 340–341, 373–374, 390,
 419, 453, 484, 486–487,
 498

 syntaxe du type de données
323 appel 127,
 180 action 123

 événement 70, 180, 268

 comme deux signaux
428 notation canonique 107, 182

cardinalité 182
changement événement 70, 183,
268 variabilité 166, 184
Cheesman, John 7
enfants 185

Christerson, Magnus 538

- classe 42, 44, 185
 - diagramme 26, 190
 - nom 192, 192 dans
 - l'état 139, 190 pris en
 - charge par le composant 218 et le type
 - 484 portée de
- classe 417 classification
 - simple et multiple 54 statique
 - et dynamique 54 classificateur
- 42, 193 rôle 194 table
 - de 43 client
 - 195 horloge
- 475
- CLOS 5, 205, 285, 390
- Coad, Pierre 5, 537
- Code
- COBOL 4
 - 295 dans les composants 216
 - et 218, génération 108
- Coleman, Derek 5, 537
- collaboration 85, 195
 - diagramme 29, 29, 89, 89, 203 et
 - diagramme de séquence 91
 - réalisation du cas d'utilisation 65
 - rôle 203
- combinaison 205
- commentaire 206
- association de communication 206
- compartiment 207
 - supplémentaire 188
 - nommé 328, 413
 - supprimant 187 temps
- de compilation 208
- contrainte
 - complète 288
- transition d'achèvement 73, 208
- transition complexe 209
 - sémantique d'exécution 445
- composant 93, 94, 216
 - diagramme 33, 222
- agrégation
 - composite 222 classe
 - 222 clé 399
 - objet 223
 - état 75, 223
 - historique état
 - dans 292 composition
- 223, 226 concret 233
- concurrency 234 sous-
 - états concurrents 210 dynamique
 - 135 envois de
 - messages 334 propriété
 - 370, 374 règles sur les
 - transitions 129
- concurrent
 - État composite 78 mot-
 - clé 370 sous-état
 - 224, 234, 463
 - sémantique d'exécution 445
 - indépendante des autres 442
 - règles pour 211
 - syntaxe des messages de thread
- 338 thread conditionnel 214, 234, 338
- contrôle de configuration 381
- conflit 234, 300
- Constantine, Larry 4, 538
- contrainte 58, 59, 101, 102, 235 langage
 - 59, 236, 271, 367 prédéfini 499
 - phase de construction
- 238 constructeur 238, 245 pour
- composite 228 conteneur
 - 238
- contexte 200, 239, 398 flux
- de
 - contrôle
 - 239 icônes 240
- Cuisinier, Steve 6

copie 243
Cox, Brad 5, 537
CRC 5
créer
 action 123
création 244, 303, 307
 événement 301,
419 notation de franchissement
de lignes 385 événement courant
 246, 301, 483 sémantique d'exécution 443

D

D'Souza, Desmond 7

notation de jetons de données 339
 type 247
 valeur 248
blocage 467
valeur par défaut 19, 249, 382
différé
 événement 139, 249,
 282 opération 390
délégation 249
DeMarco, Tom 4
dépendance 56, 58, 250
 parmi les paquets 98
 tableau de 57, 412
déploiement 252
 diagramme 34, 35, 94, 95, 252
 vue 35, 93, 254
dérivation 254
élément dérivé 254
descendant 256
descripteur 256, 304
 complet 285
Desfray, Philippe 7
design 257
 motif 387
 époque 257
DeSilva, Dilhar 7
détruisent 257
 actions 124

destruction 257
méthodes de

 développement orientées
 objet 5 traditionnelles 4

 processus 258
 extensions 531

 stéréotypes, tableau de 532
diagramme 260
symbole losange
 agrégation 148
 branche 178

 composition 228
 association n-aire 350
Digre, Tom 6
classe
 directe 261
 instance 262, 305
direction du paramètre 382
discriminateur 262
contrainte
 disjointe sur la généralisation 288 sous-
 état 264 unité
de distribution 264
Dollin, Chris 537
classe dominante 217, 219

classification dynamique 54, 106, 264, 361,
 418 concurrence 135, 140, 265, 437
 vue 9, 60, 265

E

Eddy, Frederick 5, 538
modèle efficace 471
Eiffel 5

phase d'élaboration 265
temps écoulé 475
liste d'éléments 327

ellipse dans les listes
328 document intégré 453
Embley, Brian 537

action d'entrée 74, 75, 266, 435
 sémantique d'exécution 443
énumération 267
environnement 105
Ericsson, Maria 538
événement 68, 267
 Gestion 441,
 tableau de 68
exceptions, 269,
sémantique d'exécution, 440,
action de sortie, 74, 75, 270, 435,
 sémantique d'exécution, 443
exportation
271 expression 271
extension 272, 490
 condition 274
mécanismes d'extensibilité 10, 37, 101,
 531
point d'extension 278
étendue 19, 279
transition externe 71
Eykholt, éd. 7

F

façade 503
fonctionnalité
280 stéréotype de fichier
503 filtrage de listes 329
état final 280
 sémantique d'exécution 444 feu
281 flux
90, 282 table de
 91 focus de
contrôle 283 utilisation de
polices 283 fourche
210, 284, 463 argument
formel, voir paramètre
Fortran 4
package de base 516 framework
284, 503 ami 285

gelé (mot-clé) 184 descripteur
complet 285, 299, 305, 360 vue fonctionnelle
285
Fusion 5

G

Gamma, Erich 537
collecte des déchets 228
élément généralisable 286
généralisation 51, 51, 53, 287 comparé
 à la réalisation 54 dimension de
 262 discriminateur 262

cas d'utilisation 490,
494 machine à états du générateur 448
Gery, Eran 7
Gilchrist, Helena 537 mot-
clé
 global 478
 variable 496
Goldberg, Adele 5, 538
marqueur graphique 291, 451
Griss, Martin 6, 538
propriété de groupe dans la liste
328 état de garde 72, 291, 479 gardé
(mot-clé) 370 guillemets 292

Gullekson, Garth 538

H

problème d'arrêt 467
Harel, David 7, 446, 448, 538 table de
hachage 402
Hayes, Fiona 537
Helm, Richard 537
histoire de l'état 292
 sémantique de l'exécution 446
Hogg, John 7 lien
hypertexte 293

identité 247, 294, 304, 360, 394 du
 composant 217 modèle
 mal formé 106, 294 implémentation
 295 classe 295 héritage
 295 vue
 10, 33, 93, 296
 importation 98, 296 dans
 (mot-clé) 382 inactif
 296 phase
 d'initialisation
 296 inclure 297, 491
 contrainte incomplète
 288 modèle
 294

 modèle incohérent 109
 développement incrémental 299
 valeur d'index 398
 instance indirecte 299, 305
 héritage 52, 299
 opération polymorphe 390 privé
 394 initial

 état 300
 sémantique d'exécution 444
 valeur 167, 303
 évaluation 245
 initialisation 303
 entrée/sortie (mot-clé) 382
 instance 304
 direct 261
 portée au niveau de l'instance
 417 instance de 58,
 306 instanciable
 306 instancier 306
 instanciation 307 du
 modèle 59
 intention 19, 308
 interaction 86, 308
 diagramme 309
 vue 27, 85, 310

interface 55, 310
 héritage 314
 spécificateur
 315 transition interne 75, 75, 316, 436
 sémantique d'exécution 444
 syntaxe 482
 invariant 317
 expression
 d'itération 318
 envoi de message 335
 développement itératif 258, 259, 319
 processus 433
 lyengar, Shridar 7

J

Jacobson, Agneta 538
 Jacobson, Ivar 5-7, 537-538
 Jérémie, Paul 537
 Johnson, Ralph 537
 rejoint 210, 319, 463
 Jonsson, Patrik 538
 jonction état 320

K

mot-clé 322
 dans la note
 359 prédéfini 499
 Khalsa, GK 7
 Kleppe, Anneke 538
 Kobryn, Cris 7
 Kurtz, Barry 537

L

étiquette
 322
 langage orienté objet
 5 type 323
 couche 323

feuille 286, 323
 opération 389
 ligne de vie 88,
 324 lien
 325 création
 307 fin
 327 transitoire 477
 Liskov, Barbara 52, 458 liste

 compartiment 207
 éléments 236, 327
 local (mot-clé) 478
 emplacement
 330 table de consultation 399, 402
 Lorensen, William 5, 538

M

beaucoup 331
 Martin, James 538
 Machine à farine 446
 Mellor, Stephen 4–5, 538
 membre 331

 allocation de mémoire 257
 par composite 227
 fusion 331
 notation 240

 message 90, 333
 méta-métamodèle 340

 Méta-objets 340 métaclasse
 340 métamodèle

 105, 340
 UML 515, 517

 métaobjet 340
 métarelation 340 méthode
 340 combinaison
 205, 389–391

 Meunier, Régine 537
 Meyer, Bertrand 5, 390, 538
 migration d'objets 254
 Miller, Joaquin 7

modèle 100, 342
 contenu 17
 définition 13
 élément 343
 incohérent 106, 109
 niveaux de 15
 progiciel de
 gestion 517
 vue 10, 36, 97, 343
 signification 19
 objectif 13
 aperçu de
 la modélisation
 13 temps
 344 outil
 108 module 344

 MOF, voir Meta-Object Facility
 Machine de Moore 446

 multiobjet 344

classification multiple 54, 261, 345, 361, 418
 héritage 53, 53, 287, 345

 multiplicité 167, 346
 association 348
 attribut 349
 classe 349

 association n-aire 351
 association qualifiée 400–401

N

association n-aire 350
 nom 353
 comme qualificateur
 401 espace de noms
 353, 353 navigabilité
 354 navigable
 356 navigation 356
 efficacité 356, 402
 expression 156, 339, 354, 367–368, 399,
 414–415, 415 utilisant
 des liens 325

imbrication pour la composition 229

nœud 94, 357

notation

canonique 107

considérations 106

résumé 519 note

359

valeur nulle 109

O

objet 360

création 307

diagramme 60, 363

flux 136, 139, 363

état 364

Ligne de vie

367 jouant plusieurs rôles 197

jouant le rôle 194

ensemble 123

expression définie 367

Langage de contraintes d'objets, voir le

diagramme d'objets OCL 61

Groupe de gestion des objets 6, 515, 538 orienté

objet, historique de 5

Objectif C 5

Objection 5, 8, 531

OCL 59, 124, 236, 271, 367, 409, 515

Odell, James 7, 538

Oh mon Dieu, voir Object Management Group

OMT 5, 8

opération 369

réalisation par collaboration 200

optimisation comme raffinement 409

ordonnancement 374,

374 de listes 328

sous-état orthogonal 377 out

(mot-clé) 382 Övergaard,

Gunnar 7, 538 chevauchement

(contrainte) 288 opérations prioritaires

390 portée du propriétaire 167,

377, 417

propriété 378

P

paquet 97, 99, 378

dépendance 98, 250

Palmkvist, Karin 7

paramètre 381 mot-

clé 478 liste 383

élément paramétré, voir modèle parent 383

partie,

composite 227 participe

384 objet passif 384

chemin 384 nom de

chemin 353–

354, 386 notation 193

modèle 91, 91, 387

collaboration paramétrée 197 permission

388 objet persistant

389

Jeton du réseau de Petri

464 vues physiques 32, 93

Platon 411

Politi, Michal 538

polymorphe 52, 118, 389 propriété

d'attribut 374 propriété de

polymorphisme 370, 407 postcondition 391

type de puissance 392

précondition 392

Premerlani, William 5, 537–538 élément

de présentation 393 type primitif

394 privé 394 héritage

394 contrôle

procédural 426

message 336

expression de procédure 395

- processus 395
 - extensions 531
- producteur-consommateur 464
- produit 395 profil 104
- considérations relatives au
 - langage de programmation 107
- projection 396
- propriété 396 liste 396
 - notation pour sur classe 193
 - chaîne dans liste 328
- protégé 397
- pseudo-attribut 354, 397 pseudo-état 397 public 398

Q

- association qualifiée 48, 398 efficacité
 - de navigation 357 qualificateur 398
- requête 371, 404
- propriété d'attribut 374

R

- Ramackers, Guus 7
- Rational Software Corporation 5
- Conception structurée en temps réel 4
- réalisation 55, 55, 405
 - de classe par interface 311
 - comparée à la généralisation 54 de l'opération 196 du cas d'utilisation 196, 406, 491
- récepteur 407
- réception 407 du
 - signal 427
- enregistrement comme composite 228 récurrence 123
- Reenskaug, Trygve 7

- référence 408
 - notation pour 193
- raffinement 56, 409
- réification 411 de
 - l'association 159 relation 45, 411 tableau de 46, 412 dépôt 412 requête 412 exigence 413
- suivi 477
- responsabilité 413, 413

- retour
 - action 124
 - mot-clé 382
 - message 336
 - type 371
- réutilisation 414
- rétro-ingénierie 108 minimisation des
 - risques 319
- Robson, David 5, 538
- Rohnert, Hans 537 rôle 414 nom du rôle 414
- racine 287
- Rumbaugh, James 5–7, 537–538 temps de course 416
- course jusqu'à la fin 416
 - épreuve actuelle 247

S

- scénario 416
- champ d'application
 - 371, 417 de l'opération 374
- Seidewitz, Ed 6 mot-clé
 - soi 478 langage 249 transition 418 sémantique d'exécution 443

Selic, Bran 7, 538
niveaux
 sémantiques
 45 point de variation 20, 105–107, 418
sémantique 419
envoi 333, 419
 action 124
 dépendance 423
 notation 240
expéditeur 423
séquence
 diagramme 28, 87, 87, 88 et
 diagramme de collaboration 91 numéro
 427 diagramme
de séquence 28 mot-clé
séquentiel
 370 message 336

Shlaer, Sally 5, 538
Court, Keith 7
signal 69, 427
 déclaration 69
 événement 268,
 429 notation 240
signature 429
 correspondant à
369 état
 simple 430
 transition 430
Simula-67 5

classification unique 54, 430
 hérité 430 singleton
430
Smalltalk 5, 169, 245, 249, 271, 340–341,
 374, 390, 484, 486
instantané 60, 360, 431
société d'objets coopérants 85 extensions
du processus de développement logiciel
 531

Sommerlad, Pierre 537
source
 portée 431
 état 431
spécialisation 431
 dimension de 262
spécification 432
 de l'opération 373
étapes de modélisation 432
Stal, Michael 537 état
70, 433 table de
 76 machine à
états 67, 77, 78, 439 sémantique
 d'exécution 440 règles
 d'imbrication 142
 spécification du comportement de classe
 447 spécification de l'ordre d'exécution
 447 vue 30, 67, 448
diagramme d'états 31, 448

classification statique 54, 106, 448
 vue 9, 25, 41, 60, 448
stéréotype 103, 104, 108, 449 icône
 189 notation
 dans une liste 328
 prédéfini 499 chaîne
452
caractéristique
 structurelle 453
 vue 453

Analyse structurée 4
Conception structurée 4
état stub 454
transition stubée 454 sous-
classe 455 sous-
machine 78, 436, 455 état de
 référence 456 sous-état
458 principe de
substituabilité 51, 151, 394,
 458

sous-système 100, 458
sous-type 460
résumé 460 superclasse
461 supertype 461
fournisseur 461
couloir de nage
83, 136–137, 461 barre de

synchronisation 214
 état 463
synchronisation 210 barre
 214 action
synchrone 467
 contrôle 426,
 428 message 336
 syntaxe du livre
11 système 467
stéréotype 509

T

actions
 du tableau 74
 stéréotypes de modélisation d'entreprise 534
 classificateurs
43 dépendances 57 412
stéréotypes de processus de développement 532
événements 68
flux 91
relations 46 412 vues à
 travers 39
États 76
transitions 71,
 relations de cas d'utilisation 65 277,
 vues et diagrammes 24,
balises
 467, prédéfinies 499
valeur étiquetée 102, 103, 108, 467

objet
 cible ensemble
 334 portée 168, 417, 469
 état 470
Modèle 173, 470
 instancié 174, action
de terminaison 124, thread
474, heure 474

 événement 70, 269, 474, 475
message de délai d'attente
336 marque de
synchronisation 476 jeton,
réseau de Petri 464 outil de
modélisation 108
trace 56,
 476 lien transitoire
 325, 477 objet
478 transition 71, 72, 478, 481
 complexe 209
 règles de concurrence 129
 règles de
 déclenchement
440 interne
436 phase 478 segment
442–443 table
de 71 temps
483 délai de transmission 335
notation
 arborescente
 agrégation 148
 composition 229 pour la
généralisation 289 déclencheur 479, 483
 événement 72
transition sans déclencheur 484
tuple 484
type 256, 484
 expression 168, 486 et
 classe d'implémentation 484

U

UML

- Domaines conceptuels 9
- Définition 3

- Environnement 105
- Objectifs 8 Historique 4
- Métamodèle 515, 517
- Résumé de la notation 519
- Extensions de processus 531
- Documents
 - de spécification
- 515 Normalisation 6
- Adaptation 104
- Vues 23 Tableau de 24 Débrancher, voir fusionner Diverger, voir joindre Unifié, définition de 7
- Langage de modélisation unifié (UML), voir UML
- non interprété 486 action
 - 124 valeur
- non spécifiée 109, 486 utilisation
- 56, 487 cas
- d'utilisation 488
 - création 307
 - diagramme 27, 64, 494
 - généralisation 494
 - instance 496
 - modèle 496
 - relations 66
 - tableau de 65 277
 - vue 26, 63, 496
- utilitaire 496

V

- valeur valide de l'instance système
- 60, voir la valeur des données 497
- sommet 497

- Afficher
 - 497 connexions parmi 38
 - résumés 23
 - tableau de
- 24 visibilité 379, 497
- règles 99, 119
- Vlissides, Jean 537

W

- Ward, Paul 4, 538
- Warmer, Jos 6, 538
- processus de développement de la cascade
- 433 bien formé 60, 106, 109, 211, 294, 449, 471, 498
- Wiegert, Oliver 7
- Wiener, Lauren 5, 538
- Wilkerson, Brian 5, 538
- Wirfs-Brock, Rebecca 5, 538
- Woodfield, Scott 537 travail
- en cours 109

X

- xor 156

Y

- Yourdon, Edward 4–5, 537–538

Z

- zéro, différent de l'absence de valeur
- 167